Curs 1

2019-2020

Programare Logică

Cuprins

Organizare

Instructori

Curs:

- **Ioana Leuștean**

Seminar/Laborator

- **Carmen Chiriță**
- **Alexandru Dragomir**
- **Ioana Leuștean**
- **Ana Țurlea**

Suport curs/seminar/laborator

- https://cs.unibuc.ro//~ileustean/PL.html
- http://moodle.fmi.unibuc.ro/course/view.php?id=186

O parte din materiale sunt realizate în colaborare cu Denisa Diaconescu.

Notare

- **Laborator: 30 puncte**
- **Examen: 60 puncte**
- Se acordă 10 puncte din oficiu!

Notare

- **Laborator: 30 puncte**
- **Examen: 60 puncte**
- Se acordă 10 puncte din oficiu!

- Condiție minimă pentru promovare:
  laborator: minim 15 puncte și
  examen: minim 25 puncte.

Notare

- **Laborator: 30 puncte**
- **Examen: 60 puncte**
- Se acordă 10 puncte din oficiu!

- Condiție minimă pentru promovare:
  laborator: minim 15 puncte și
  examen: minim 25 puncte.

- Se poate obține punctaj suplimentar pentru activitatea din timpul seminarului/laboratorului:
  maxim 10 puncte.

Laborator: 30 puncte

Testare:

- Are loc în ultimele două săptămâni.
- Data concretă o să fie anunțată ulterior.
- Prezența este obligatorie pentru a putea promova!
- Pentru a trece această probă, trebuie să obțineți minim 15 puncte.
- Un student va fi absent numai daca este absent la ambele probe.

Examen: 60 puncte

- Subiecte de tip exercițiu:
  - în stilul exemplelor de la curs;
  - în stilul exercițiilor rezolvate la seminarii și laboratoare.
- Timp de lucru: 2 ore
- Pentru a trece această probă, trebuie să obțineți minim 25 puncte.
- Materiale ajutătoare la examen: slide-urile printate și legate.

Curs/seminar/laborator

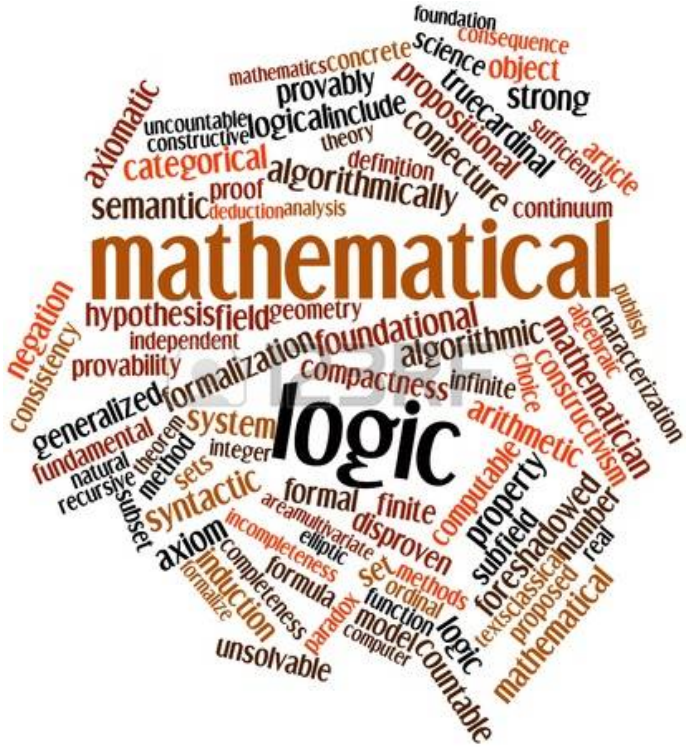
- Curs
  - Programare logică
    - Deducția naturală
    - Logica clauzelor Horn
    - Unificare
    - Rezoluție
  - Semantică și verificarea programelor
    - implementarea semanticii unui mini-limbaj în Prolog
  - Topici speciale (selecție)
    - sisteme de rescriere, programare ecuațională
    - elemente avansate de programare Prolog
- Seminar
  - Exerciții suport pentru curs.
- Laborator: Prolog
  - cel mai cunoscut limbaj de programare logică
  - bazat pe logica clauzelor Horn
  - semantica operațională este bazată pe rezoluție

Bibliografie

- M. Ben-Ari, **Mathematical Logic for Computer Science**, Springer, 2012.
- P. Blackburn, J. Bos, K. Striegnitz, **Learn Prolog now**, College Publications, 2006.
- M. Huth, M. Ryan, **Logic in Computer Science: Modelling and Reasoning about Systems**, Cambridge University Press New York, 2004.
- J.W. Lloyd, Foundations of Logic Programming, Springer, 1987.
- Logic Programming, The University of Edinburgh, https://www.inf.ed.ac.uk/teaching/courses/lp/

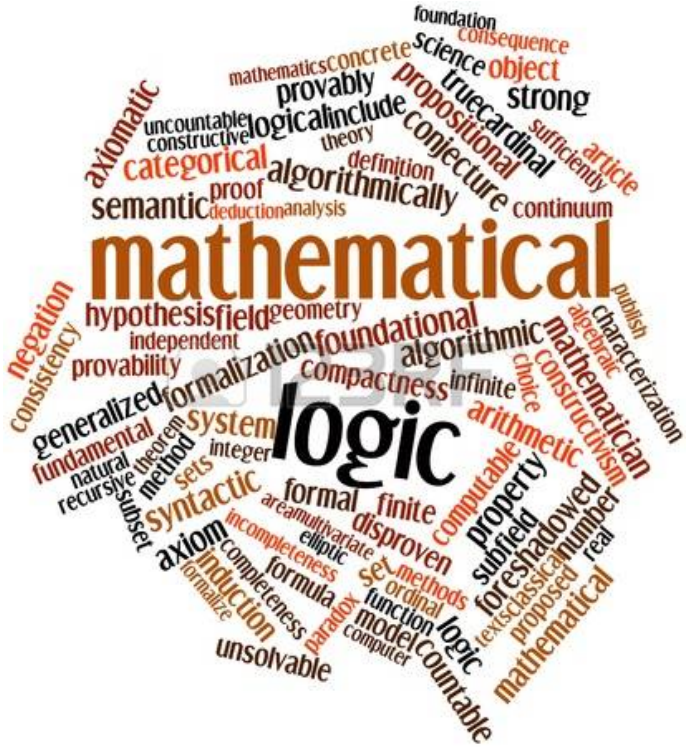
Logica matematică

- Un mijloc de a clarifica/modela procesul de a "raționa".
- Logica ne permite să reprezentăm/modelăm probleme.
- Care logică?
  - propozițională
  - de ordinul I
  - de ordin înalt
  - logici modale
  - logici temporale
  - logici cu mai multe valori
  - ...



Logica matematică

- Un mijloc de a clarifica/modela procesul de a "raționa".
- Logica ne permite să reprezentăm/modelăm probleme.
- Care logică?
  - propozițională
  - de ordinul I
  - de ordin înalt
  - logici modale
  - logici temporale
  - logici cu mai multe valori
  - ...



La acest curs, veți vedea cum poate fi folosită logica în programare și în verificarea programelor.

Privire de ansamblu

Principalele paradigme de programare

- Imperativă (cum calculăm)
  - Procedurală
  - Orientată pe obiecte
- Declarativă (ce calculăm)
  - Logică
  - Funcțională

Principalele paradigme de programare

- Imperativă (cum calculăm)
  - Procedurală
  - Orientată pe obiecte
- Declarativă (ce calculăm)
  - Logică
  - Funcțională

La acest curs, veți invăța programare logică.

Programare declarativă

- Programatorul spune ce vrea să calculeze, dar nu specifică concret cum calculează.
- Este treaba interpretorului (compilator/implementare) să identifice cum să efectueze calculul respectiv.
- Programarea logică este un tip de programare declarativă!
- Tipuri de programare declarativă:
  - Programare logică (e.g., Prolog)
  - Programare funcțională (e.g., Haskell)

Programare logică

- Programarea logică este o paradigmă de programare bazată pe logică formală.

Programare logică

- Programarea logică este o paradigmă de programare bazată pe logică formală.
- Unul din sloganurile programării logice:

  Program = Logică + Control *(R. Kowalski)*

Programare logică

- Programarea logică este o paradigmă de programare bazată pe logică formală.
- Unul din sloganurile programării logice:

  Program = Logică + Control *(R. Kowalski)*

- Programarea logică poate fi privită ca o deducție controlată.

Programare logică

- Programarea logică este o paradigmă de programare bazată pe logică formală.
- Unul din sloganurile programării logice:

  Program = Logică + Control *(R. Kowalski)*

- Programarea logică poate fi privită ca o deducție controlată.
- Un program scris într-un limbaj de programare logică este

  o listă de formule într-o logică

  ce exprimă fapte și reguli despre o problemă.

Programare logică

- Programarea logică este o paradigmă de programare bazată pe logică formală.
- Unul din sloganurile programării logice:

  Program = Logică + Control *(R. Kowalski)*

- Programarea logică poate fi privită ca o deducție controlată.
- Un program scris într-un limbaj de programare logică este

  o listă de formule într-o logică

  ce exprimă fapte și reguli despre o problemă.
- Exemple de limbaje de programare logică:
  - Prolog
  - Answer set programming (ASP)
  - Datalog

Ce veți vedea la laborator

Prolog

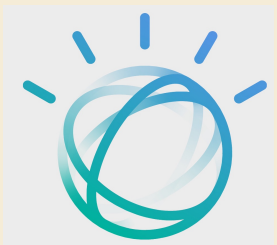
- bazat pe logica clauzelor Horn
- semantica operațională este bazată pe rezoluție
- este Turing complet
- vom folosi implementarea SWI-Prolog

Ce veți vedea la laborator

Prolog

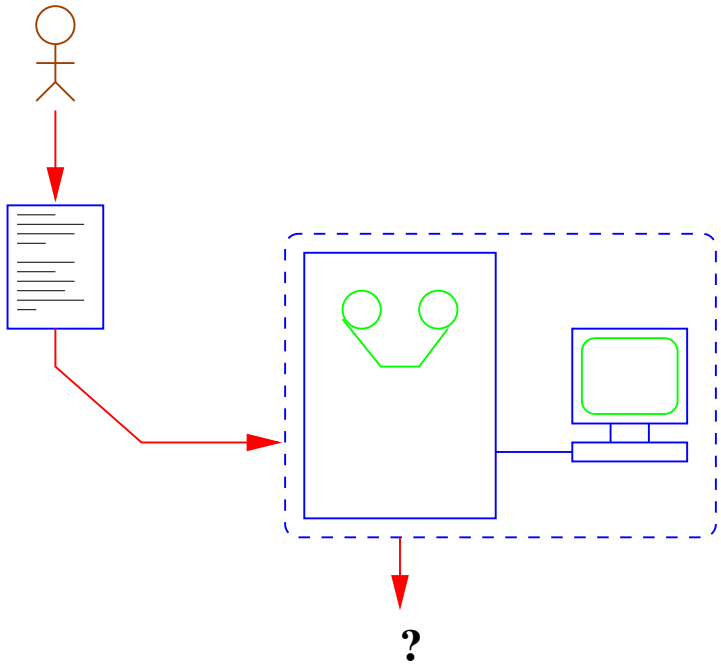
- bazat pe logica clauzelor Horn
- semantica operațională este bazată pe rezoluție
- este Turing complet
- vom folosi implementarea SWI-Prolog

Limbajul Prolog este folosit pentru programarea sistemului IBM Watson!



Puteți citi mai multe detalii aici.

Problema corectitudinii programelor



- Pentru anumite metode de programare (e.g., imperativă, orientată pe obiecte), nu este ușor să stabilim că un program este corect sau să înțelegem ce înseamnă că este corect (e.g, în raport cu ce?!).
- Corectitudinea programelor devine o problemă din ce în ce mai importantă, nu doar pentru aplicații "safety-critical".
- Avem nevoie de metode ce asigură "calitate", capabile să ofere "garanții".

Un program imperativ simplu

```
#include <iostream>
using namespace std;
int main()
{
  int square;
  for(int i = 1; i <= 5; ++i)
  {
    square = i * i;
    cout <<  square << endl;
  }
}
```

Un program imperativ simplu

```
#include <iostream>
using namespace std;
int main()
{
  int square;
  for(int i = 1; i <= 5; ++i)
  {
    square = i * i;
    cout <<  square << endl;
  }
}
```

- Este corect?

Un program imperativ simplu

```
#include <iostream>
using namespace std;
int main()
{
  int square;
  for(int i = 1; i <= 5; ++i)
  {
    square = i * i;
    cout <<  square << endl;
  }
}
```

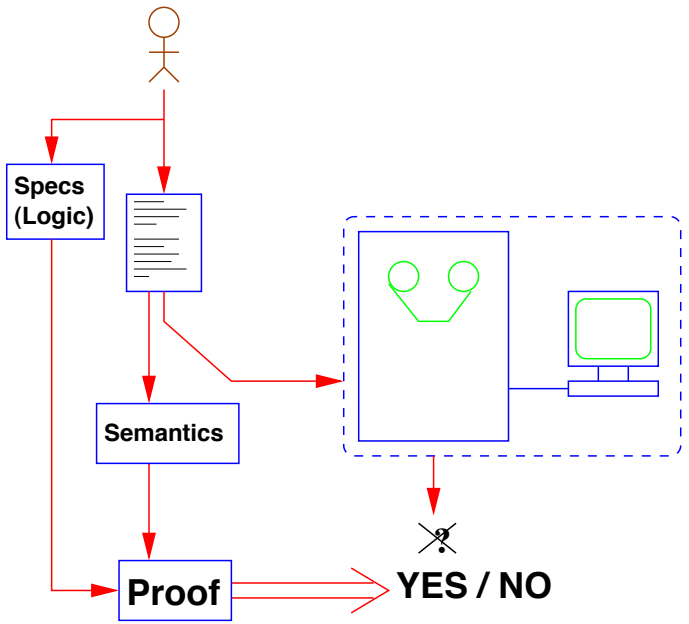
- Este corect? În raport cu ce?

Un program imperativ simplu

```
#include <iostream>
using namespace std;
int main()
{
  int square;
  for(int i = 1; i <= 5; ++i)
  {
    square = i * i;
    cout <<  square << endl;
  }
}
```

- Este corect? În raport cu ce?
- Un formalism adecvat trebuie:
  - să permită descrierea problemelor (specificații), și
  - să raționeze despre implementarea lor (corectitudinea programelor).

Corectitudinea programelor



Pentru a scrie specificații și a raționa despre corectitudinea programelor:

- Limbaje de specificații (modelarea problemelor)
- Semantica programelor (operațională, denotațională, ...)
- Demonstrații (verificarea programelor, ...)

Tipuri de semantică

Semantica dă "înțeles" unui program.

Tipuri de semantică

Semantica dă "înțeles" unui program.

- Operațională:
  - Înțelesul programului este definit în funcție de pașii (transformări dintr-o stare în alta) care apar în timpul execuției.

Tipuri de semantică

Semantica dă "înțeles" unui program.

- Operațională:
  - Înțelesul programului este definit în funcție de pașii (transformări dintr-o stare în alta) care apar în timpul execuției.
- Denotațională:
  - Înțelesul programului este definit abstract ca element dintr-o structură matematică adecvată.

Tipuri de semantică

Semantica dă "înțeles" unui program.

- Operațională:
  - Înțelesul programului este definit în funcție de pașii (transformări dintr-o stare în alta) care apar în timpul execuției.
- Denotațională:
  - Înțelesul programului este definit abstract ca element dintr-o structură matematică adecvată.
- Axiomatică:
  - Înțelesul programului este definit indirect în funcție de axiomele și regulile pe care le verifică.

Tipuri de semantică

Semantica dă "înțeles" unui program.

- Operațională:
  - Înțelesul programului este definit în funcție de pașii (transformări dintr-o stare în alta) care apar în timpul execuției.
- Denotațională:
  - Înțelesul programului este definit abstract ca element dintr-o structură matematică adecvată.
- Axiomatică:
  - Înțelesul programului este definit indirect în funcție de axiomele și regulile pe care le verifică.

La acest curs, vom defini un limbaj și semantica lui folosind Prolog.

Programare logică & Prolog

Programare logică - în mod idealist

- Un "program logic" este o colecție de proprietăți presupuse (sub formă de formule logice) despre lume (sau mai degrabă despre lumea programului).
- Programatorul furnizează și o proprietate (o formula logică) care poate să fie sau nu adevărată în lumea respectivă (întrebare, *query*).
- Sistemul determină dacă proprietatea aflată sub semnul întrebării este o consecință a proprietăților presupuse în program.
- Programatorul nu specifică metoda prin care sistemul verifică dacă întrebarea este sau nu consecință a programului.

Exemplu de program logic

$$
\begin{aligned}
\texttt{oslo} &\rightarrow \texttt{windy} \\
\texttt{oslo} &\rightarrow \texttt{norway} \\
\texttt{norway} &\rightarrow \texttt{cold} \\
\texttt{cold} \wedge \texttt{windy} &\rightarrow \texttt{winterIsComing} \\
&\quad\ \texttt{oslo}
\end{aligned}
$$

Exemplu de program logic

$$\begin{aligned} \texttt{oslo} &\rightarrow \texttt{windy} \\ \texttt{oslo} &\rightarrow \texttt{norway} \\ \texttt{norway} &\rightarrow \texttt{cold} \\ \texttt{cold} \wedge \texttt{windy} &\rightarrow \texttt{winterIsComing} \\ & \quad\ \texttt{oslo} \end{aligned}$$

Exemplu de întrebare

Este adevărat `winterIsComing`?

Putem să testăm în SWI-Prolog

Program:

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

Intrebare:

```
?- winterIsComing.
true
```

http://swish.swi-prolog.org/

Sintaxă: constante, variabile, termeni compuși

- Atomi: `sansa, 'Jon Snow', jon_snow`
- Numere: `23, 23.03,-1`

  Atomii și numerele sunt constante.
- Variabile: `X, Stark, _house`
- Termeni compuși: `father(eddard, jon_snow)`,
  `and(son(bran,eddard), daughter(arya,eddard))`
  - forma generală: atom(termen,. . ., termen)
  - atom-ul care denumește termenul se numește functor
  - numărul de argumente se numește aritate

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

Program în Prolog = bază de cunoștințe

Exemplu

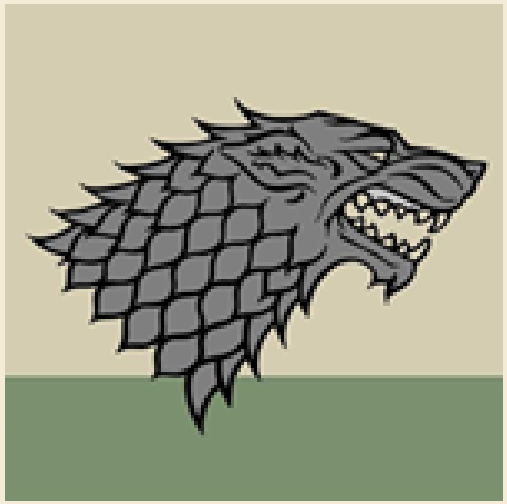
Un program în Prolog:

```
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).
```



Un program în Prolog este o bază de cunoștințe (Knowledge Base).

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.

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).
```

Predicate:
father/2
mother/2
stark/1

Un program în Prolog

Program

- Un program în Prolog este format din reguli de forma

```
Head :- Body.
```

- Head este un predicat, iar Body este o secvență de predicate separate prin virgulă.
- Regulile fără `Body` se numesc fapte.

Program

- Un program în Prolog este format din reguli de forma

```
Head :- Body.
```

- Head este un predicat, iar Body este o secvență de predicate separate prin virgulă.
- Regulile fără `Body` se numesc fapte.

Exemplu

- Exemplu de regulă: `stark(X) :- father(Y,X), stark(Y).`
- Exemplu de fapt: `father(eddard, jon_snow).`

Interpretarea din punctul de vedere al logicii

- operatorul `:-` este implicația logică $\leftarrow$

Exemplu

```
winterfell(X) :- stark(X)
```

dacă `stark(X)` este adevărat, atunci `winterfell(X)` este adevărat.

Interpretarea din punctul de vedere al logicii

- operatorul `:-` este implicația logică $\leftarrow$

Exemplu

`winterfell(X) :- stark(X)`

dacă `stark(X)` este adevărat, atunci `winterfell(X)` este adevărat.

- virgula `,` este conjuncția $\wedge$

Exemplu

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

dacă `father(Y,X)` și `stark(Y)` sunt adevărate,
atunci `stark(X)` este adevřat.

Interpretarea din punctul de vedere al logicii

- mai multe reguli cu același `Head` definesc același predicat, între defiții fiind un sau logic.

Exemplu

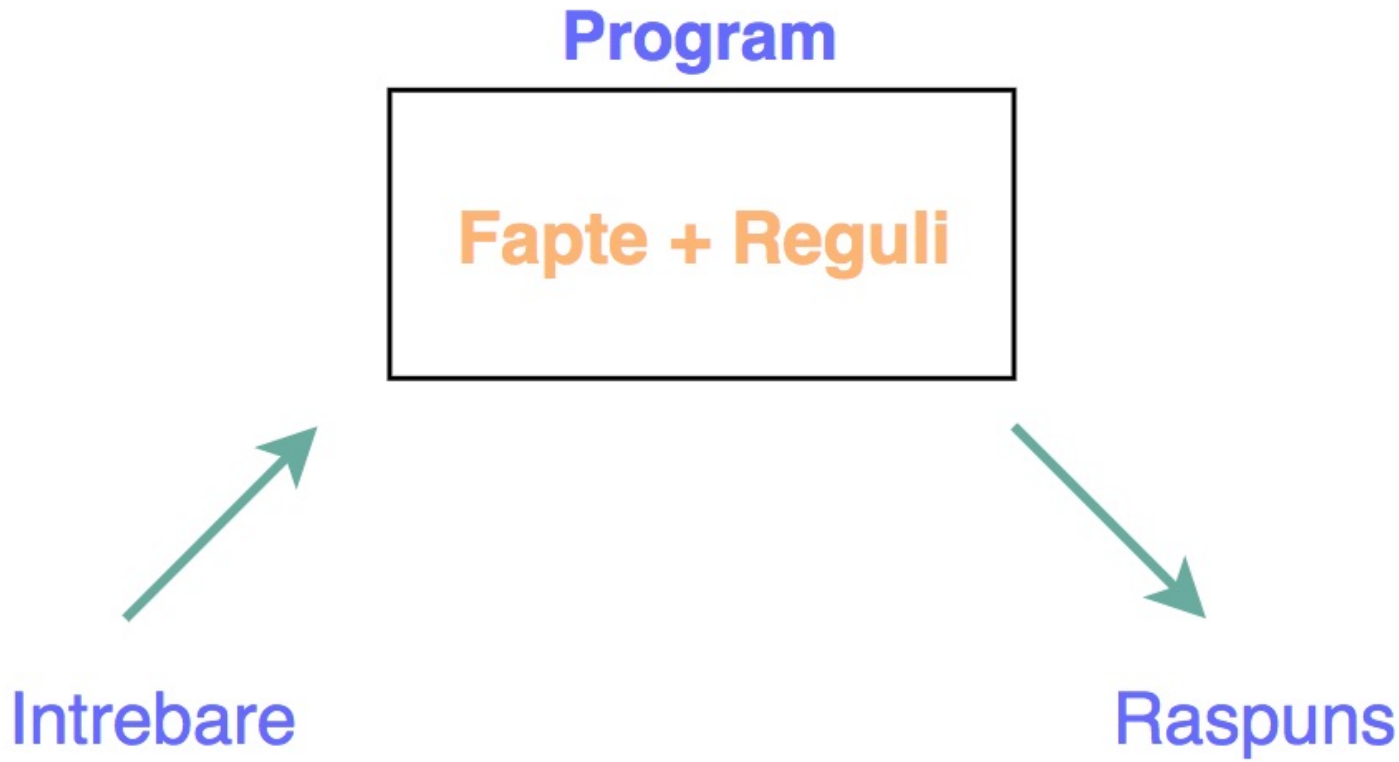
```
got_house(X) :- stark(X).
got_house(X) :- lannister(X).
got_house(X) :- targaryen(X).
got_house(X) :- baratheon(X).
```

dacă
`stark(X)` este adevărat sau
`lannister(X)` este adevărat sau
`targaryen(X)` este adevărat sau
`baratheon(X)` este adevărat,
atunci
`got_house(X)` este adevărat.

Un program în Prolog

Cum folosim un program în Prolog?

Întrebări în Prolog



Întrebări și ținte în Prolog

- Prolog poate răspunde la întrebări legate de consecințele relațiilor descrise într-un program în Prolog.
- Întrebările sunt de forma:

  $$\texttt{?- predicat}_1\texttt{(...),...,predicat}_n\texttt{(...).}$$

- Prolog verifică dacă întrebarea este o consecință a relațiilor definite în program.
- Dacă este cazul, Prolog caută valori pentru variabilele care apar în întrebare astfel încât întrebarea să fie o consecință a relațiilor din program.
- un predicat care este analizat pentru a se răspunde la o întrebare se numește țintă (goal).

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.

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.

Exemplu

```
?- stark(jon_snow)
true
?- stark(wylla)
false
```

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

Cum găsește Prolog răspunsul

Pentru a găsi un raspuns, 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.
```

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`.

Vom discuta detaliat algoritmul de unificare!

Cum găsește Prolog răspunsul

Pentru a găsi un raspuns, 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
```

Dacaă nu se poate face potrivirea, răspunsul este false.

Cum găsește Prolog răspunsul

Pentru a găsi un raspuns, 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 raspuns, 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, 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.

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 raspuns, Prolog redenumește variabilele.

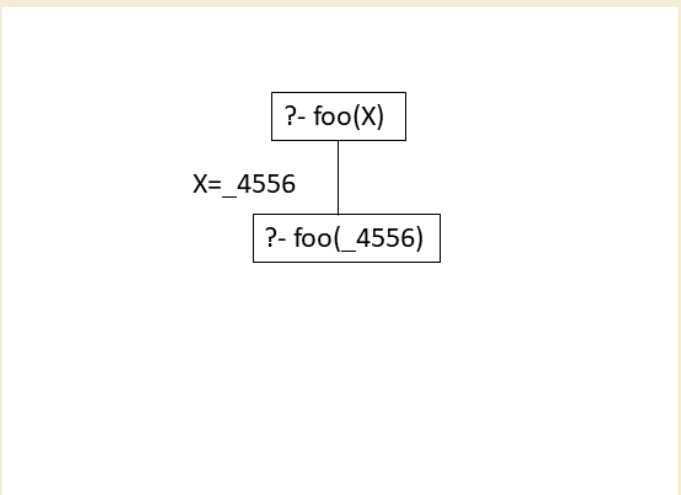
Exemplu

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 raspuns, Prolog încearcă regulile în ordinea apariției lor.

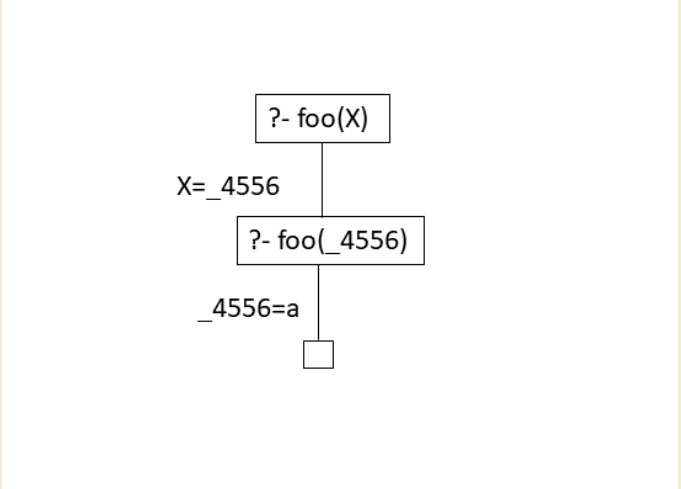
Exemplu

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 raspuns, Prolog încearcă clauzele în ordinea apariției lor.

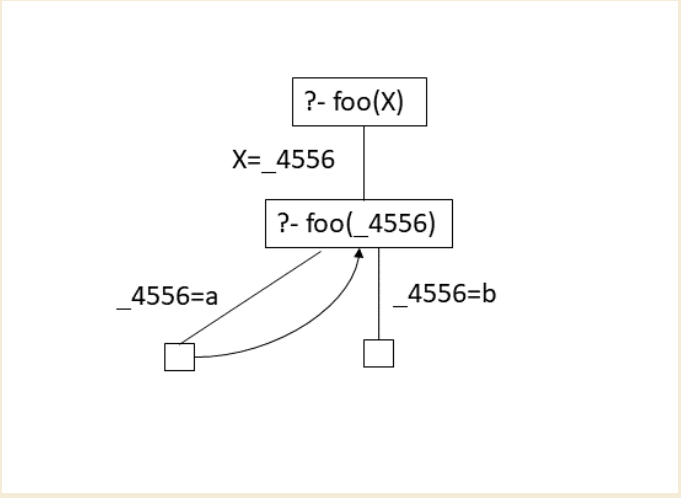
Exemplu

Să presupunem că avem programul:

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

și că punem următoarea întrebare:

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



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.

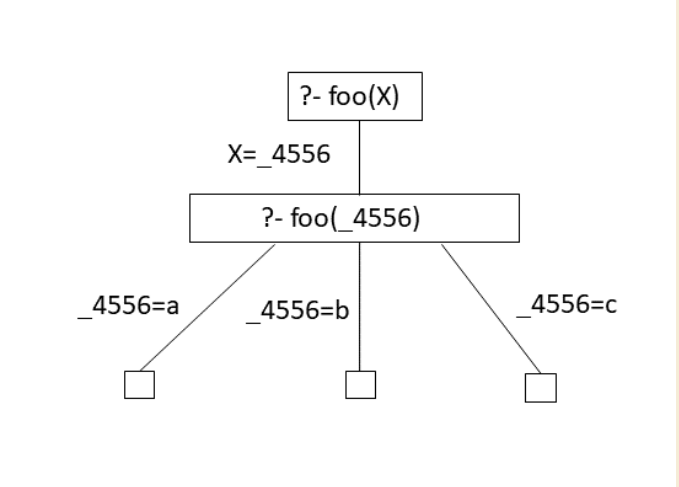
Exemplu

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

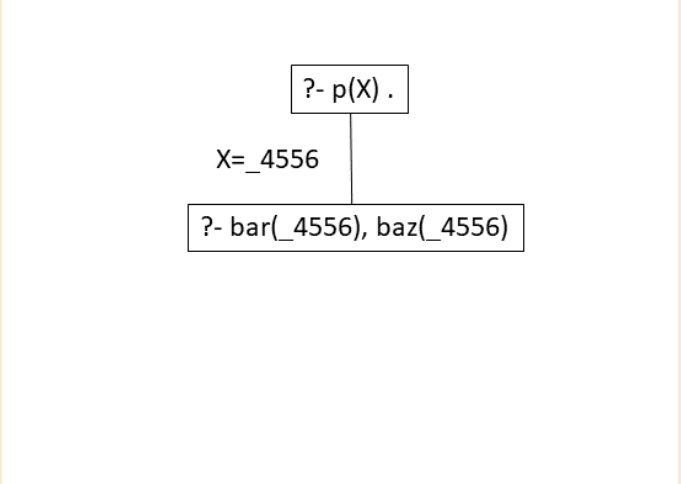
Să presupunem că avem programul:

```
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.

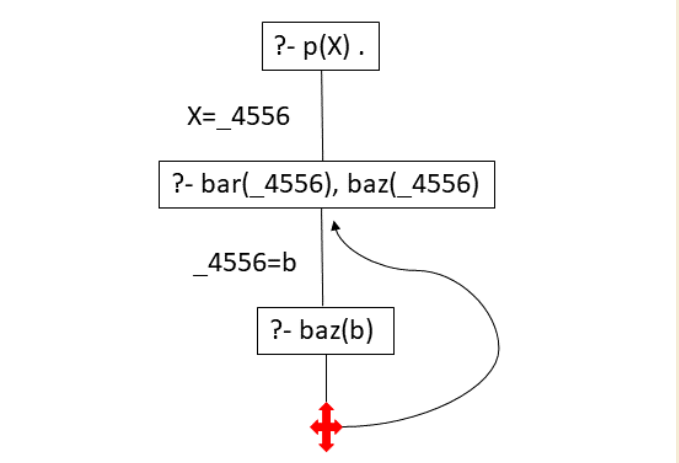
Exemplu

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.

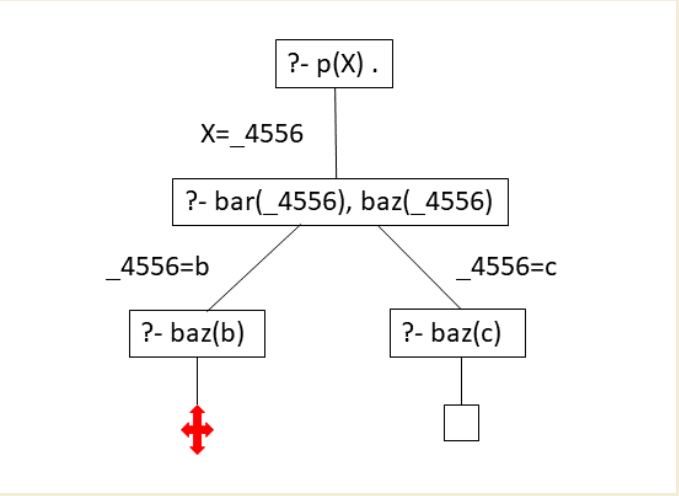
Exemplu

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
```

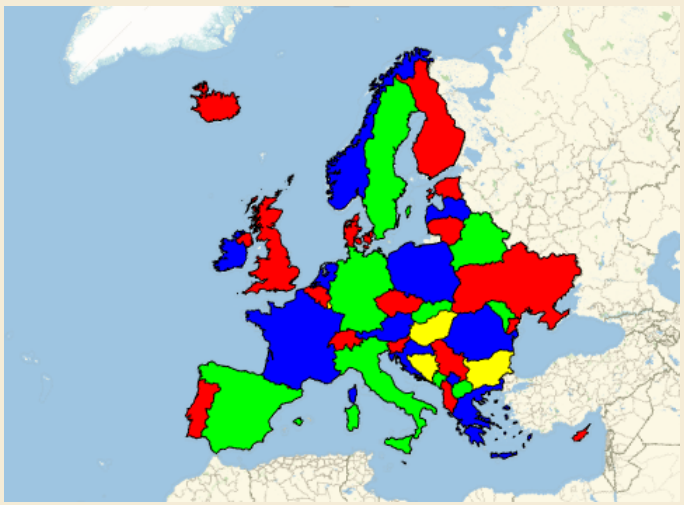
Vă explicați ce s-a întâmplat? Desenați arborele de căutare!

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

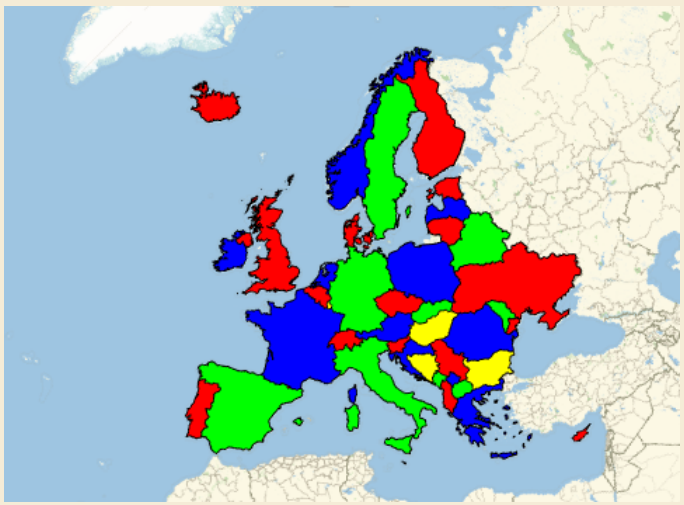
Un program mai complicat

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

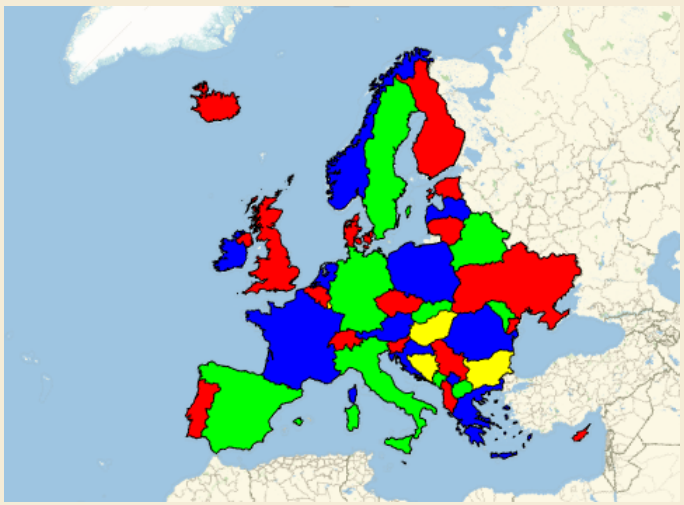
Să se coloreze o hartă dată cu un număr minim de culori astfel încât oricare două țări vecine să fie colorate diferit.

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 \= 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

Exemplu

```
?- 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.

Pe săptămâna viitoare!