Curs 1

2017-2018

Programare Logică

Cuprins

1 Organizare

2 Privire de ansamblu

3 Programare logică & Prolog

Organizare

Instructori

Curs:

- **Denisa Diaconescu** (seria 23)
- **Ioana Leuștean** (seria 24)

Seminar:

- **Carmen Chiriță** (grupele 241, 242)
- **Denisa Diaconescu** (grupa 235)
- **Alexandru Dragomir** (grupele 244, 311)
- **Bogdan Dumitru** (grupa 243)
- **Ana Țurlea** (grupele 231, 232, 233, 234)

Instructori

Laborator:

- **Cosmin-Silvian Alexandru** (grupa 231)
- **Carmen Chiriță** (grupele 234, 241, 242)
- **Denisa Diaconescu** (grupa 235)
- **Alexandru Dragomir** (grupele 232, 244, 311)
- **Bogdan Dumitru** (grupa 243)
- **Ana Țurlea** (grupele 233)

Suport curs

- Seria 23
  - http://old.unibuc.ro/~ddiaconescu/2018/pl
  - http://moodle.fmi.unibuc.ro/course/view.php?id=494
- Seria 24
  - http://old.unibuc.ro/~ileustean/PL
  - http://moodle.fmi.unibuc.ro/course/view.php?id=186

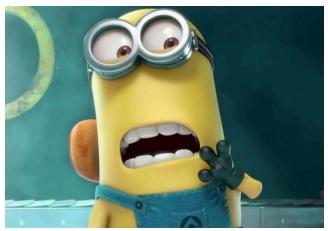
Notare

- **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!

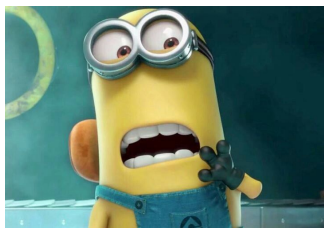
- Condiție minimă pentru promovare:
  laborator: minim 15 puncte și
  examen: minim 25 puncte.



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 Săptămâna 11 (7 – 13 mai).
- Data concretă o să fie anunțată ulterior.
- Prezența este obligatorie!
- Pentru a trece această probă, trebuie să obțineți minim 15 puncte.

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
- Aveți voie doar cu materialele de la curs printate.
- Pentru a trece această probă, trebuie să obțineți minim 25 puncte.

Curs/seminar/laborator

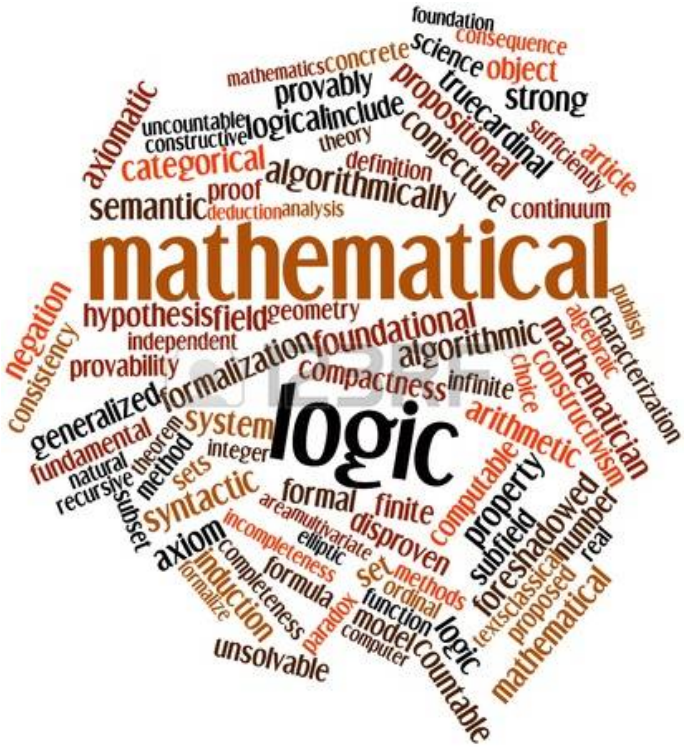
- Curs
  - Programare logică
    - Deducția naturală
    - Logica clauzelor Horn
    - Unificare
    - Rezoluție
  - Elemente de demonstrare automată
    - Sisteme de rescriere
  - Semantică și verificarea programelor
    - Logica Hoare
- Seminar
  - Exerciții suport pentru curs.
- Laborator: Prolog
  - 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.
- F. Baader, T. Nipkow, **Terms Rewriting and All That**, Cambridge University Press, 1998.

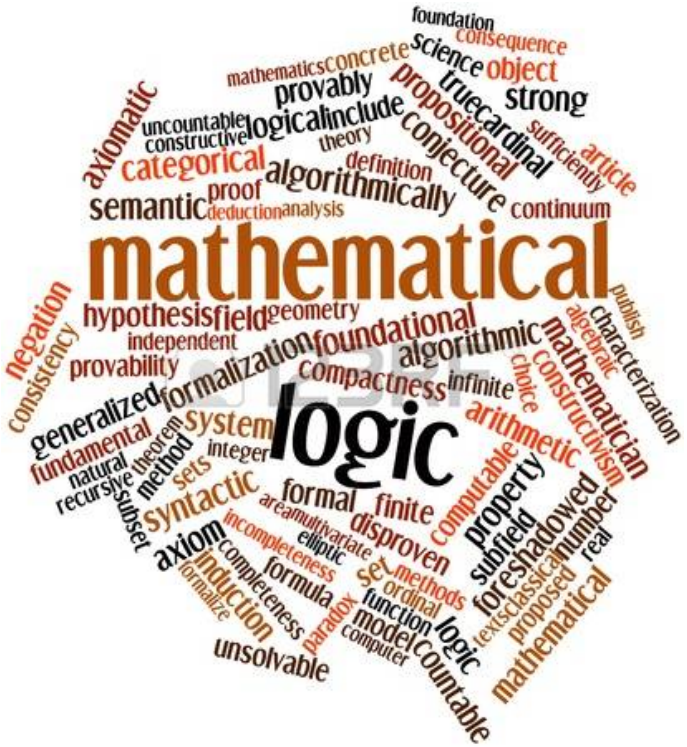
Logica matematică

- 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.

La acest curs vom folosi litere grecești:

Αα	Ββ	Γγ	Δδ	Εε	Ζζ
ALPHA [a] *ἄλφα*	**BETA** [b] *βῆτα*	**GAMMA** [g] *γάμμα*	**DELTA** [d] *δέλτα*	**EPSILON** [e] *ἒ ψιλόν*	**ZETA** [dz] *ζῆτα*
Ηη	Θθ	Ιι	Κκ	Λλ	Μμ
ETA [ɛː] *ἦτα*	**THETA** [t^h] *θῆτα*	**IOTA** [i] *ἰῶτα*	**KAPPA** [k] *κάππα*	**LAMBDA** [l] *λάμβδα*	**MU** [m] *μῦ*
Νν	Ξξ	Οο	Ππ	Ρρ	Σσς
NU [n] *νῦ*	**XI** [ks] *ξεῖ*	**OMICRON** [o] *ὂ μικρόν*	**PI** [p] *πεῖ*	**RHO** [r] *ῥῶ*	**SIGMA** [s] *σῖγμα*
Ττ	Υυ	Φφ	Χχ	Ψψ	Ωω
TAU [t] *ταῦ*	**UPSILON** [ʉ] *ὖ ψιλόν*	**PHI** [p^h] *φεῖ*	**CHI** [k^h] *χεῖ*	**PSI** [ps] *ψεῖ*	**OMEGA** [ɔː] *ὦ μέγα*

Privire de ansamblu

Principalele paradigme de programare

- Imperativă
  - Procedurală
  - Orientată pe obiecte
  - ...
- Declarativă
  - Logică
  - Funcțională
  - ...

Principalele paradigme de programare

- Imperativă
  - Procedurală
  - Orientată pe obiecte
  - ...
- Declarativă
  - 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

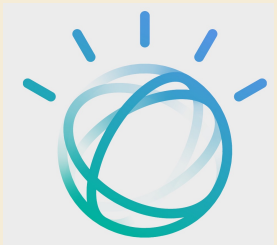
- bazat pe logica clauzelor Horn
- semantica operațională este bazată pe rezoluție
- este Turing complet
- vom folosi implementarea SWI-Prolog

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 și aici.

Sisteme de rescriere

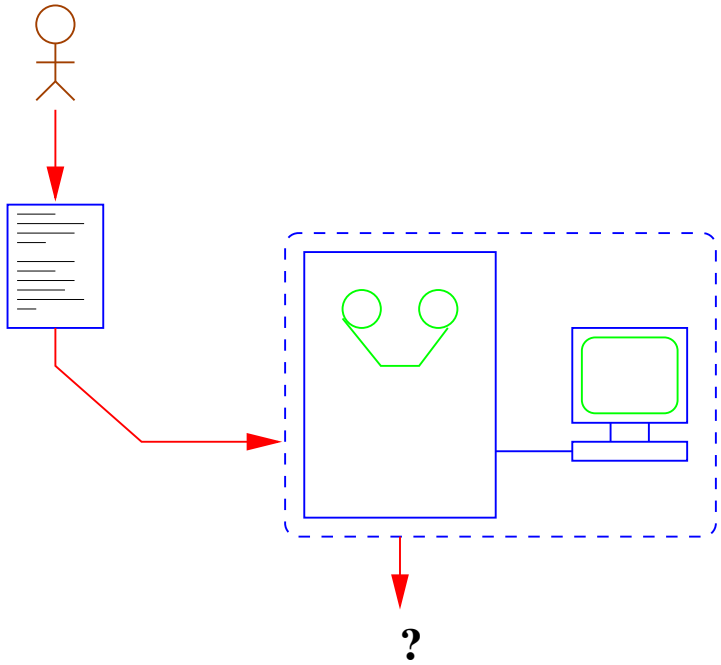
- Sistemele de rescriere modelează tranzițiile între obiecte:
  - o derivare într-o gramatică este un sistem de rescriere,
  - o mașină Turing poate fi modelată ca un sistem de rescriere,
  - un program este un sistem de tranziții între stările sistemului.
- Problema terminării unui sistem de rescriere este nedecidabilă, dar poate fi analizată cu mijloace specifice.
- Algoritmul Knuth-Bendix, bazat pe rescriere, este o metodă de demonstrare automată.

Sisteme de rescriere

- Sistemele de rescriere modelează tranzițiile între obiecte:
  - o derivare într-o gramatică este un sistem de rescriere,
  - o mașină Turing poate fi modelată ca un sistem de rescriere,
  - un program este un sistem de tranziții între stările sistemului.
- Problema terminării unui sistem de rescriere este nedecidabilă, dar poate fi analizată cu mijloace specifice.
- Algoritmul Knuth-Bendix, bazat pe rescriere, este o metodă de demonstrare automată.

Acest curs conține o introducere în teoria sistemelor de rescriere abstracte.

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;
  }
}
```

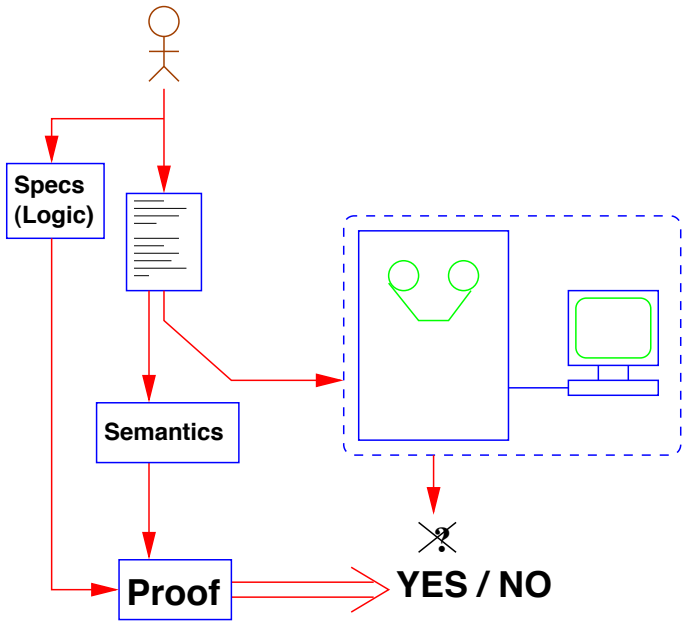
- Este corect? În raport cu ce?

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, veți intra în contact
cu toate aceste tipuri de semantici.


LET'S GET STARTED

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.

Programare logică - în mod idealist

Aspecte declarative ale programării logice:

- Programatorul nu specifică metoda prin care sistemul verifică dacă întrebarea este sau nu consecință a programului.
- Faptul că întrebarea chiar este sau nu consecință este independent de metoda aleasă de sistem.

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:

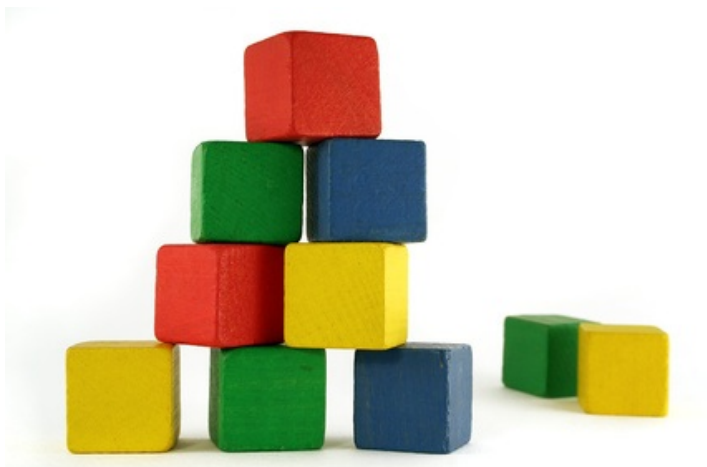
```
?- winterIsComing.
true
```

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

Un program în Prolog

Termeni

- Termenii sunt unitățile de bază prin care Prolog reprezintă datele.
- Sunt de 3 tipuri:
  - Constante: `23, sansa, 'Jon Snow'`
  - Variabile: `X, Stark, _house`
  - Termeni compuși (predicate): `father(eddard, jon_snow)`



Un mic exercițiu

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

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

- Un program în Prolog este format din reguli de forma

```
Head :- Body.
```

- `Head` este adevărat dacă `Body` este adevărat.
- Regulile fără `Body` se numesc fapte.

Program

- Un program în Prolog este format din reguli de forma

```
Head :- Body.
```

- `Head` este adevărat dacă `Body` este adevărat.
- Regulile fără `Body` se numesc fapte.

Exemplu

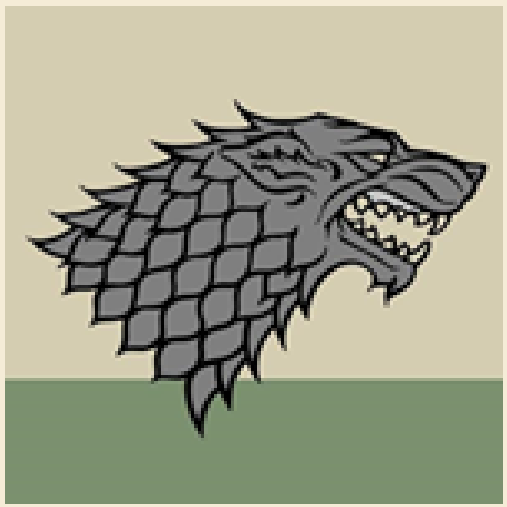
- Exemplu de regulă: `stark(X) :- father(Y,X), stark(Y).`
- Exemplu de fapt: `father(eddard, jon_snow).`

Program

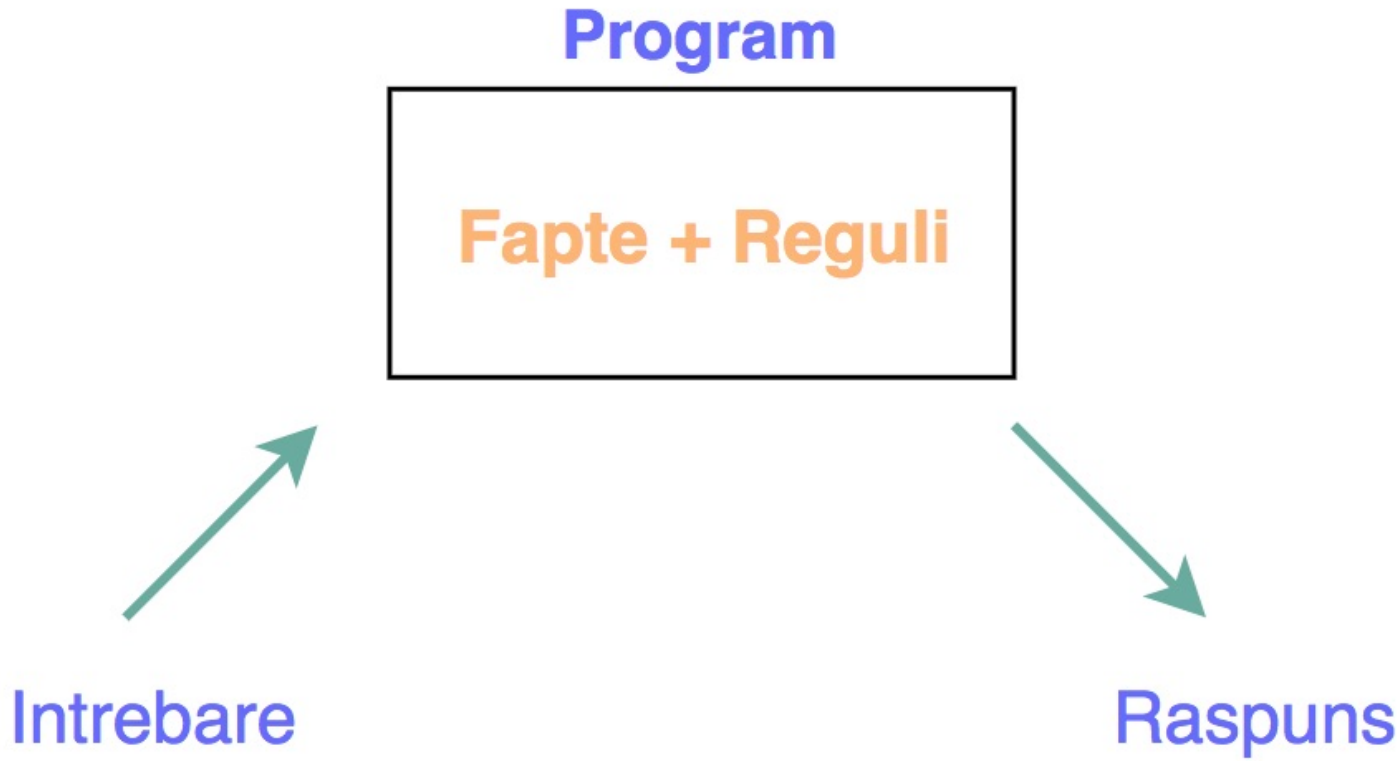
Exemplu

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).
```



Întrebări în Prolog



Întrebări 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.

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 clauzele în ordinea apariției lor.

Cum găsește Prolog răspunsul

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

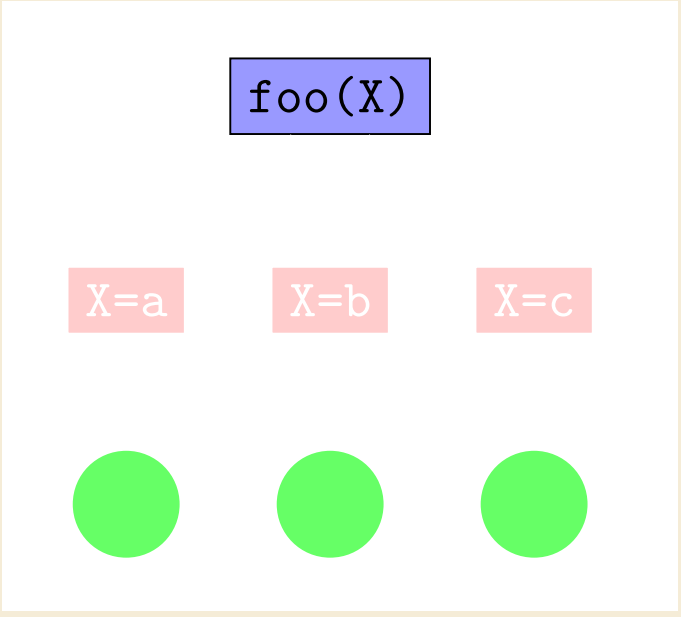
Exemplu

Să presupunem că avem programul:

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

și că punem următoarea întrebare:

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



Cum găsește Prolog răspunsul

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

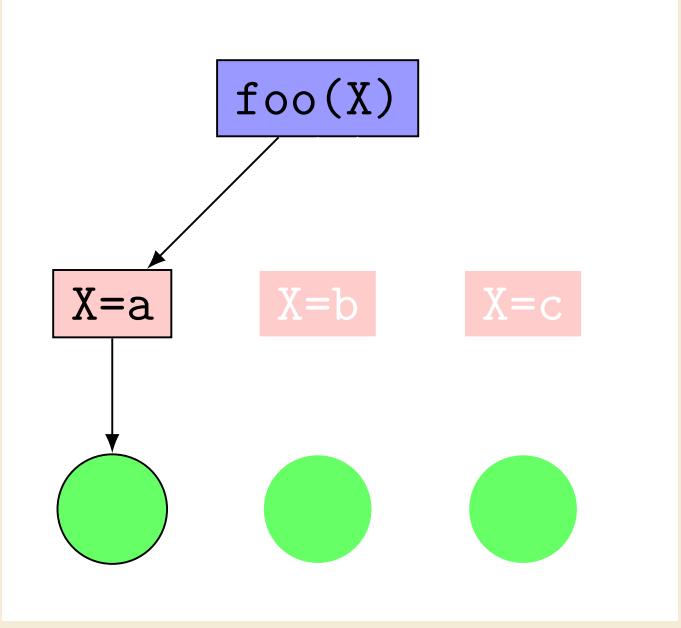
Exemplu

Să presupunem că avem programul:

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

și că punem următoarea întrebare:

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



Cum găsește Prolog răspunsul

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

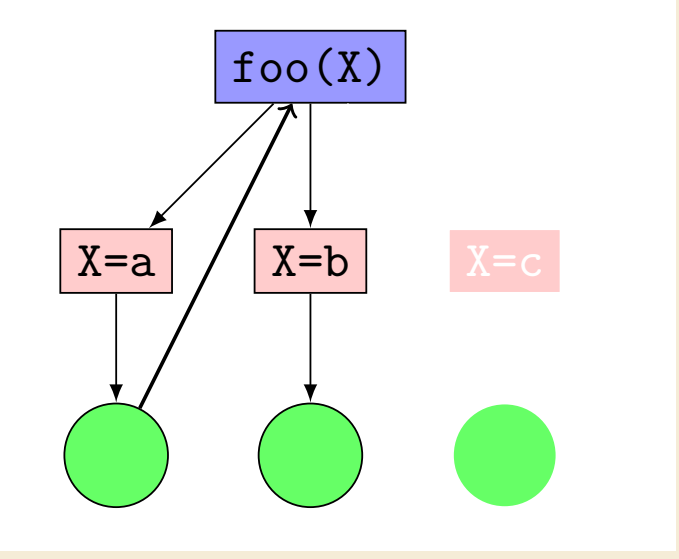
Exemplu

Să presupunem că avem programul:

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

și că punem următoarea întrebare:

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



Cum găsește Prolog răspunsul

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

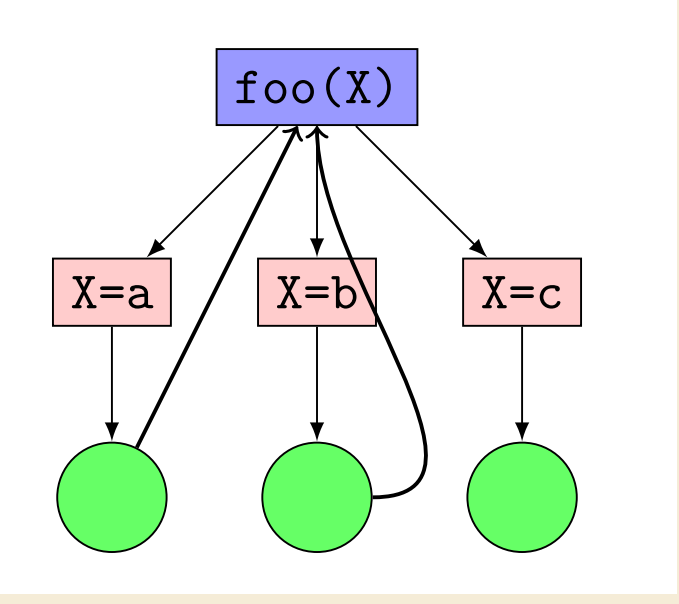
Exemplu

Să presupunem că avem programul:

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

și că punem următoarea întrebare:

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



Cum găsește Prolog răspunsul

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

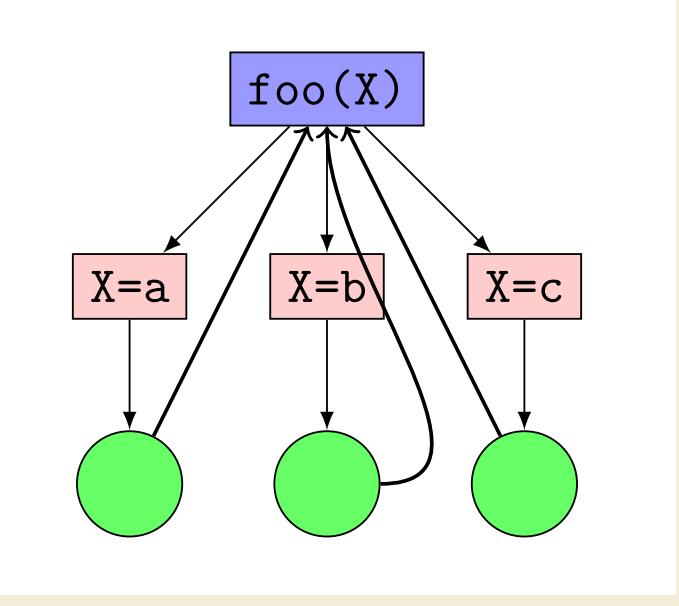
Exemplu

Să presupunem că avem programul:

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

și că punem următoarea întrebare:

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



Cum găsește Prolog răspunsul

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

Cum găsește Prolog răspunsul

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

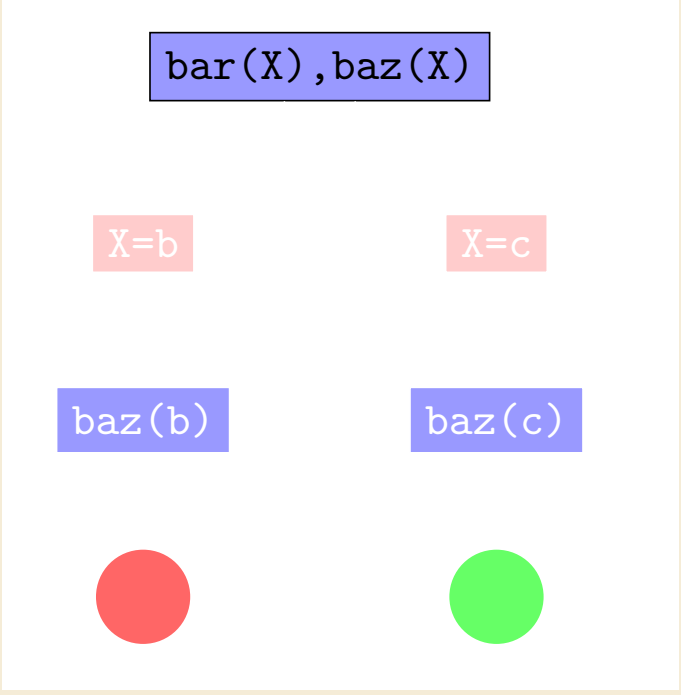
Exemplu

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.

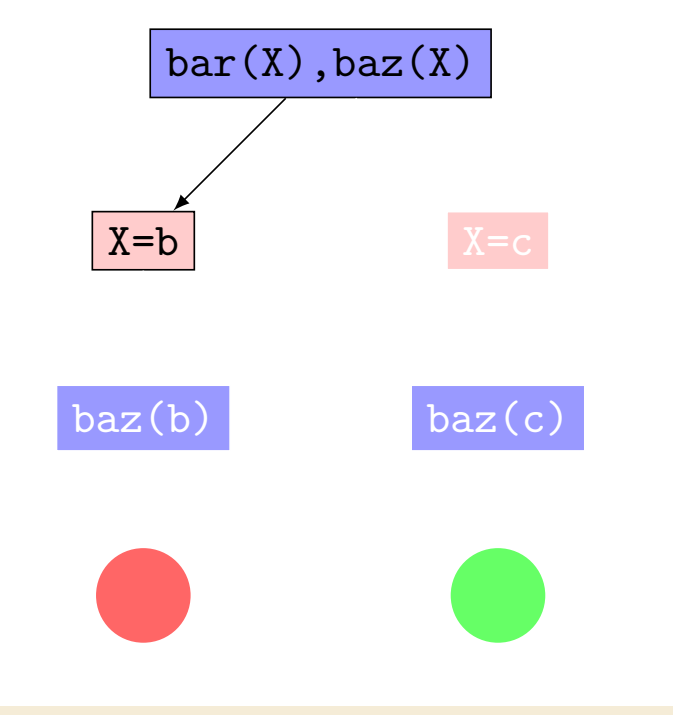
Exemplu

Să presupunem că avem programul:

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

și că punem următoarea întrebare:

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



Cum găsește Prolog răspunsul

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

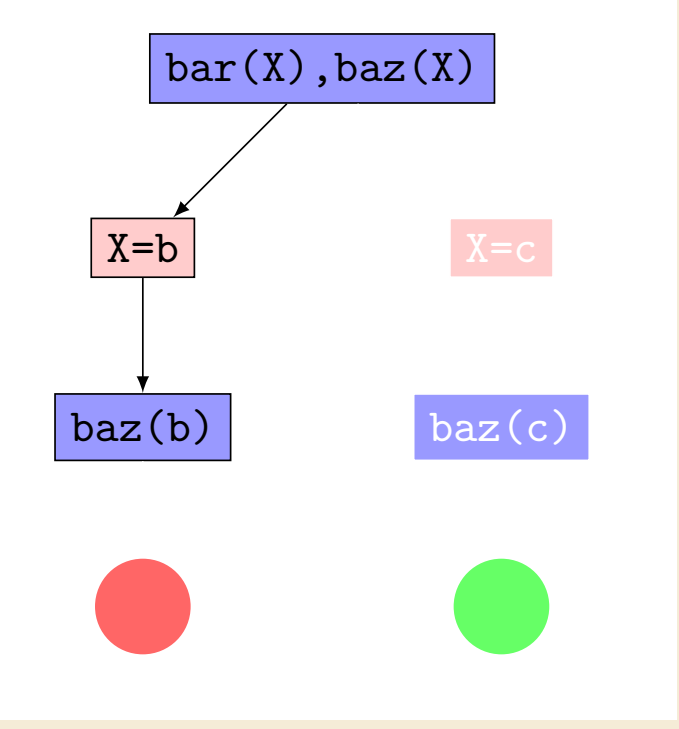
Exemplu

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.

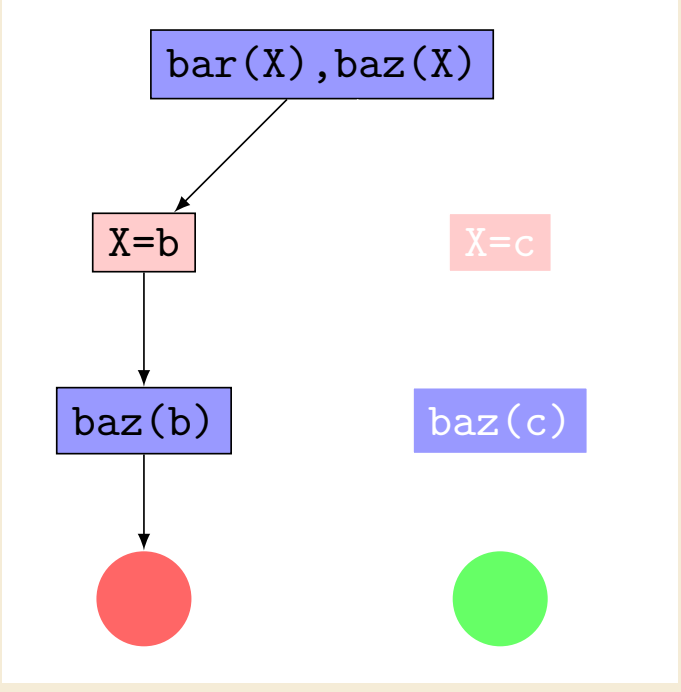
Exemplu

Să presupunem că avem programul:

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

și că punem următoarea întrebare:

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



Cum găsește Prolog răspunsul

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

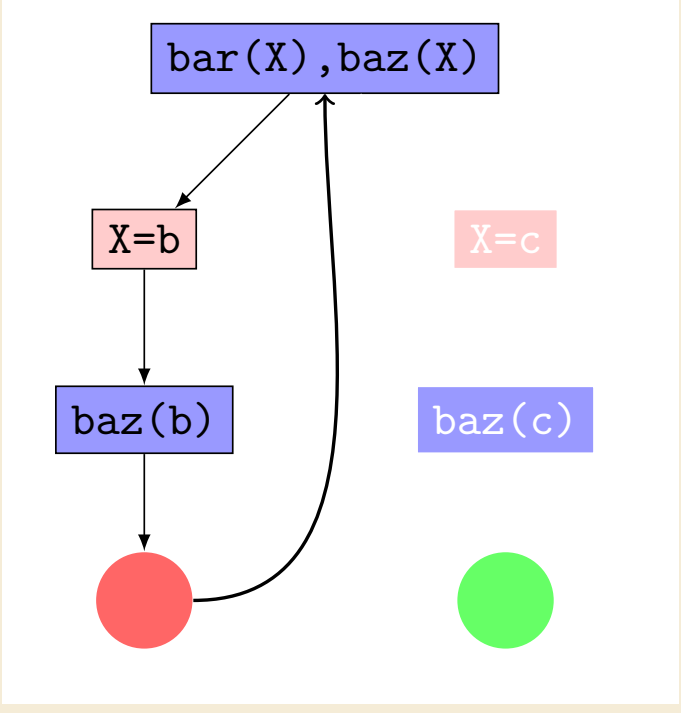
Exemplu

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.

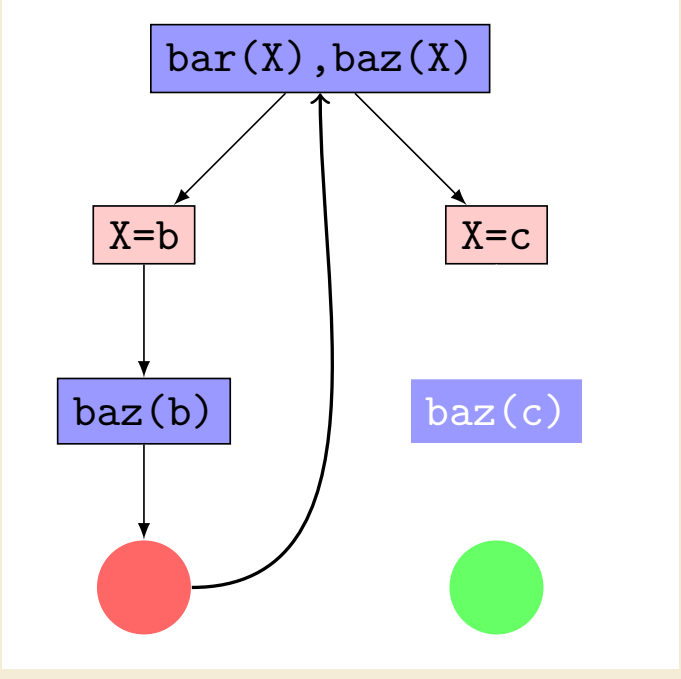
Exemplu

Să presupunem că avem programul:

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

și că punem următoarea întrebare:

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



Cum găsește Prolog răspunsul

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

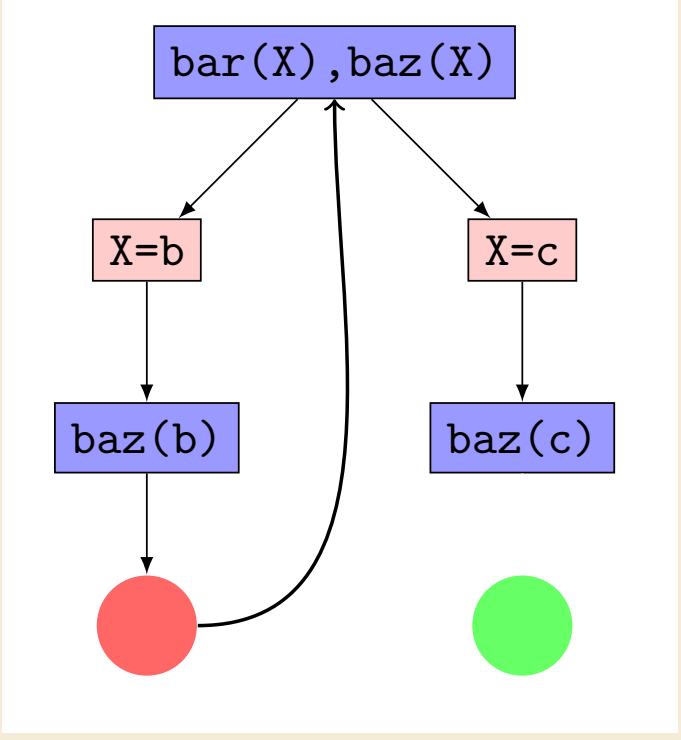
Exemplu

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.

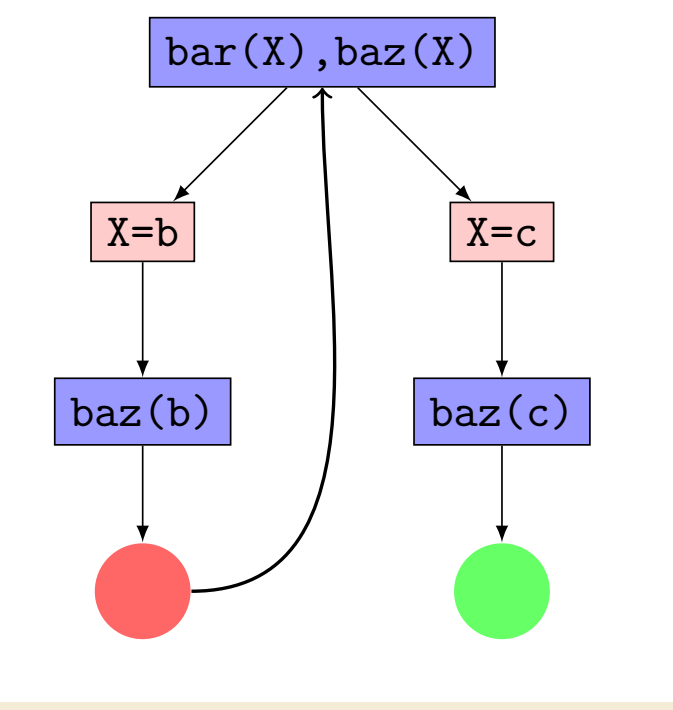
Exemplu

Să presupunem că avem programul:

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

și că punem următoarea întrebare:

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



Cum găsește Prolog răspunsul

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

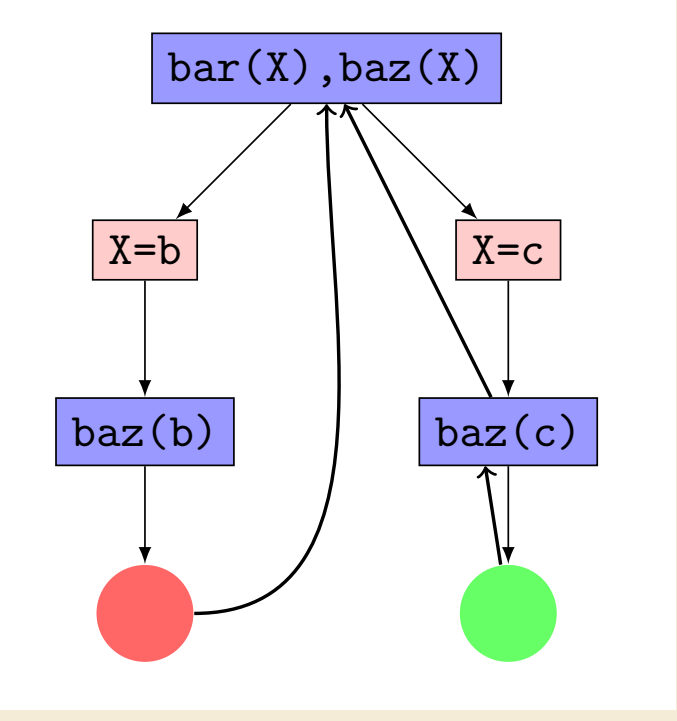
Exemplu

Să presupunem că avem programul:

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

și că punem următoarea întrebare:

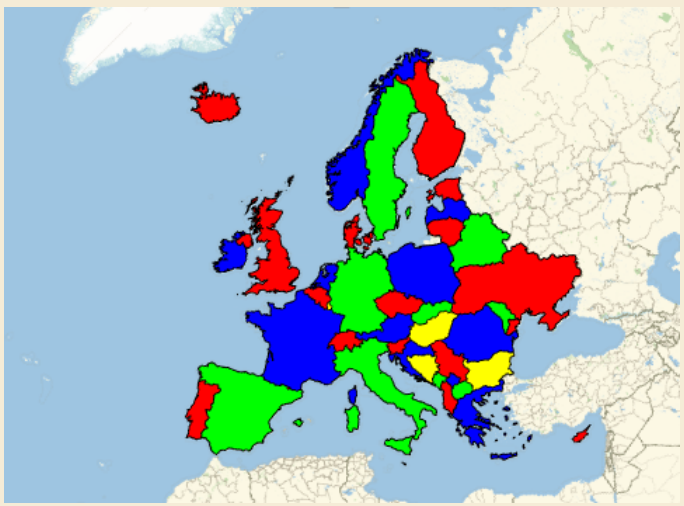
```
?- bar(X),baz(X).
```



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



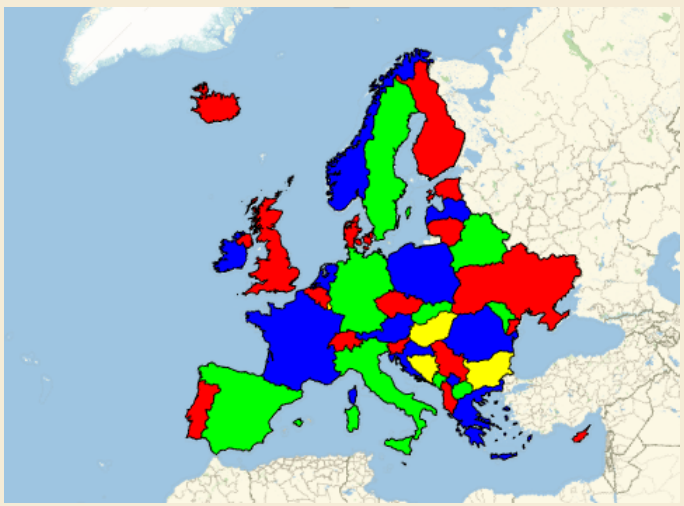
Sursa imaginii

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

Problema colorării hărților

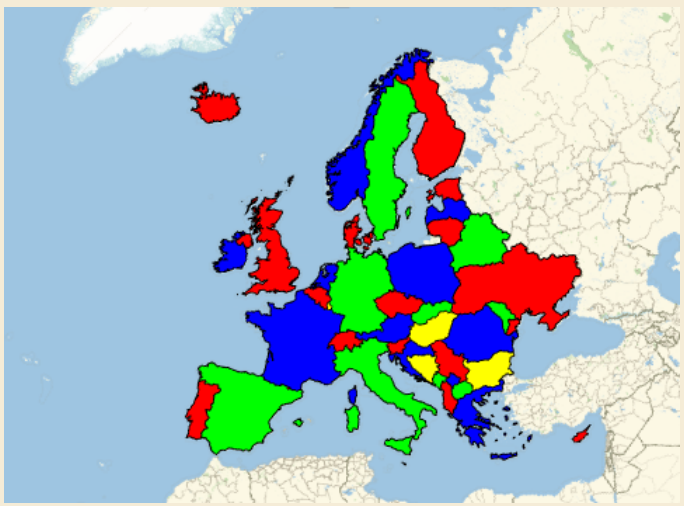
Să se coloreze o hartă dată cu un număr minim de culori astfel încât oricare două țări vecine să fie colorate diferit.

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
```

Puncte-cheie

- De ce acestea sunt răspunsurile corecte?

Puncte-cheie

- De ce acestea sunt răspunsurile corecte?
- Cum găsește formal Prolog răspunsurile?

Puncte-cheie

- De ce acestea sunt răspunsurile corecte?
- Cum găsește formal Prolog răspunsurile?
- Găsește mereu Prolog răspunsurile corecte?

Pe săptămâna viitoare!