Laborator 1

Logică matematică și computațională

Prolog

- Prolog este cel mai cunoscut limbaj de programare logică.
  - bazat pe logica clasică de ordinul I (cu predicate)
  - funcționează pe bază de unificare, rezoluție și *backtracking*
- Multe implementări îl transformă în limbaj de programare matur
  - I/O, operații implementate deja în limbaj etc.

Prolog

- Vom folosi implementarea SWI-Prolog
  - gratuit
  - folosit des cu scop pedagogic
  - conține multe biblioteci
  - `http://www.swi-prolog.org/`
- Varianta online SWISH a SWI-Prolog
  - `http://swish.swi-prolog.org/`

Prolog

Întrebări:

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

Mai multe detalii

- `Capitolul 1` din *Learn Prolog Now!*.

Sintaxă: atomi

Atomi:

- secvențe de litere, numere și _, care încep cu o literă mică
- șiruri între apostrofuri `'Atom'`
- anumite simboluri speciale

Exemplu

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

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

`atom/1` este un predicat predefinit

Sintaxă: constante și variabile

Constante:

- atomi: a, 'I am an atom'
- numere: 2, 2.5, -33

Variabile:

- secvențe de litere, numere și _, care încep cu o literă mare sau cu _
- Variabilă specială: _ este o variabilă anonimă
  - două apariții ale simbolului _ sunt variabile diferite
  - este folosită când nu vrem să folosim variabila respectivă

Exemplu

- `X`
- `Animal`
- `_x`
- `X_1_2`

Sintaxă: termeni compuși

Termeni compuși:

- au forma `p(t1,...,tn)` unde
  - `p` este un atom,
  - `t1,...,tn` sunt termeni.

Exemplu

- `is_bigger(horse,X)`
- `is_bigger(horse,dog)`
- `f(g(X,_),7)`

- Un termen compus are
  - un nume (functor): `is_bigger` în `is_bigger(horse, X)`
  - o aritate (numărul de argumente): 2 în `is_bigger(horse, X)`

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 (întrebare, *query*).
- Sistemul determină dacă proprietatea aflată sub semnul întrebării este o consecință a proprietăților presupuse în program.

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

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

Exemplu de întrebare

Este adevărat `winterIsComing`?

Exemplul de mai sus în SWI-Prolog

Program:

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

Întrebare:

```
?- winterIsComing.
true
```

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

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(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 este 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

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

Interpretarea din punctul de vedere al logicii

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

Exemplu

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

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

Interpretarea din punctul de vedere al logicii

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

Exemplu

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

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

- virgula , este 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` definesc același predicat, între definiții fiind 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,

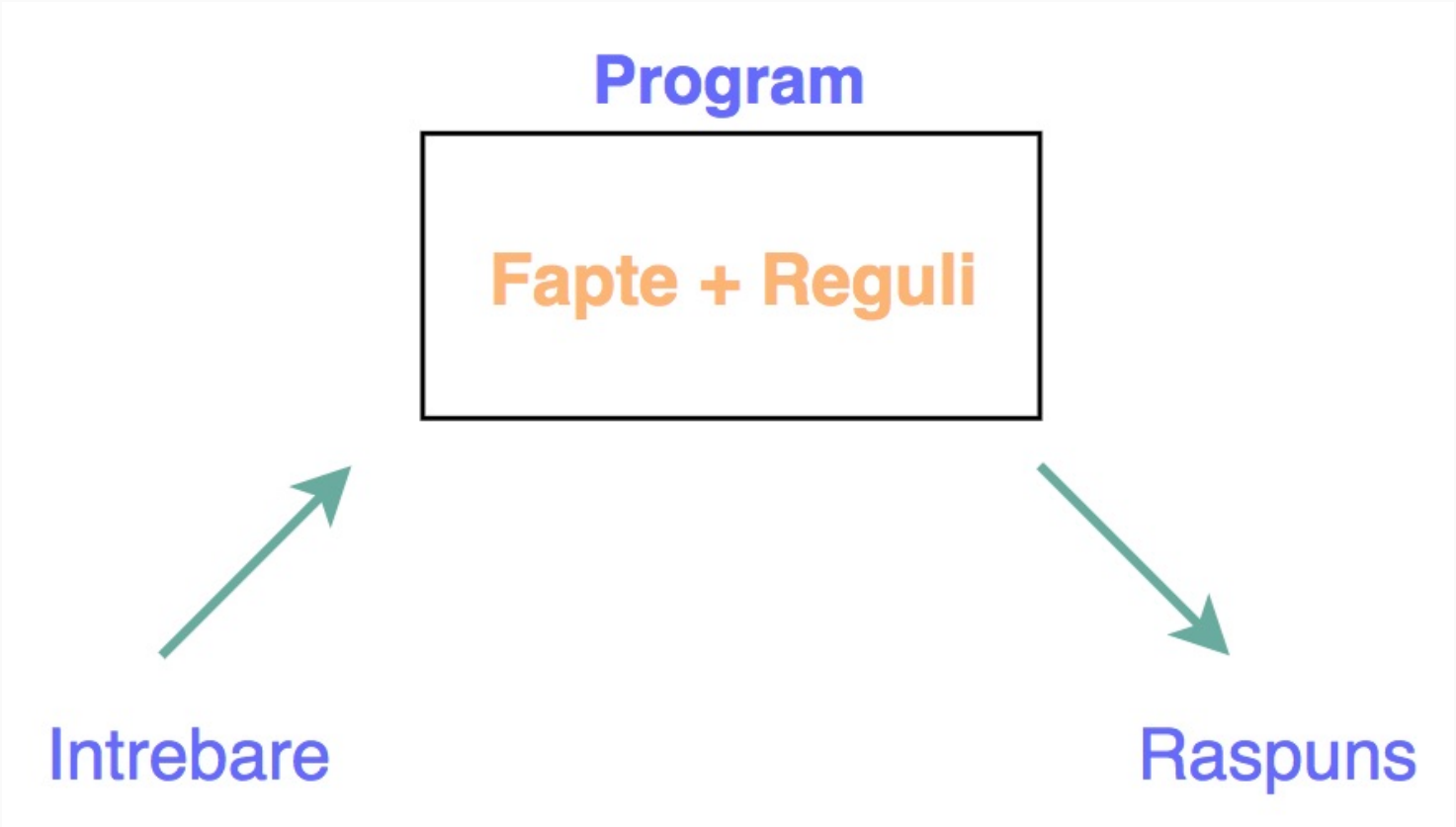
atunci

`comedy(X)` este adevărat.

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 instanțierea întrebării cu acele valori 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

```
?- 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 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.
```

Pentru a răspunde la întrebare se caută o potrivire (unificator) între scopul `foo(X)` și baza de cunoștințe. Răspunsul 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 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, tastăm `;`

```
?- 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 redenumește variabilele.

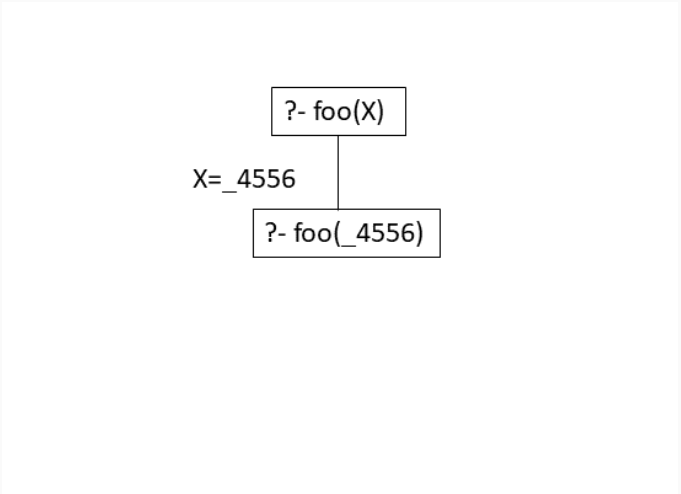
Exemplu

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 răspuns, Prolog încearcă regulile în ordinea apariției lor.

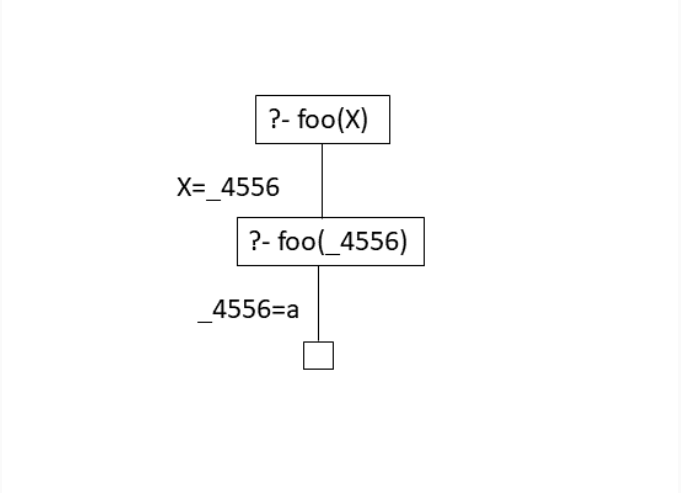
Exemplu

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 încearcă clauzele în ordinea apariției lor.

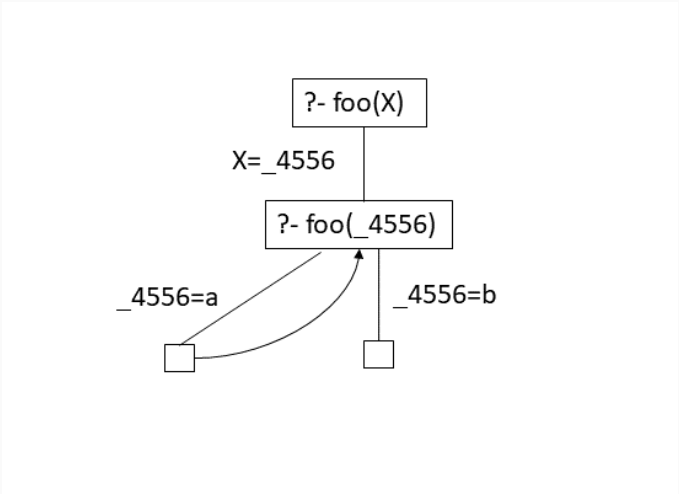
Exemplu

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.

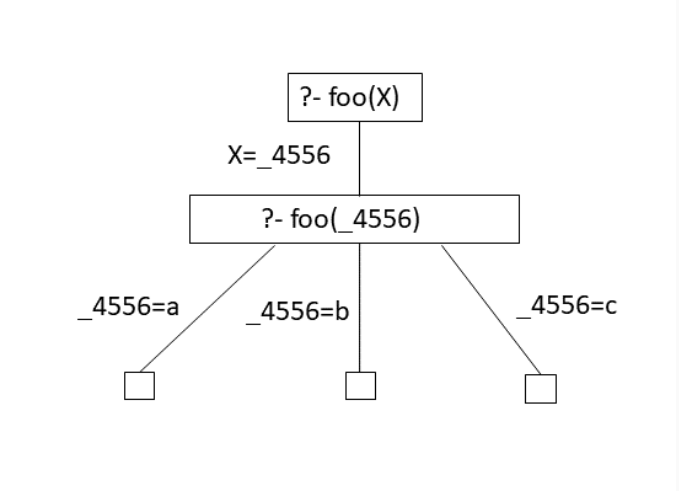
Cum găsește Prolog răspunsul

Pentru a găsi un 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

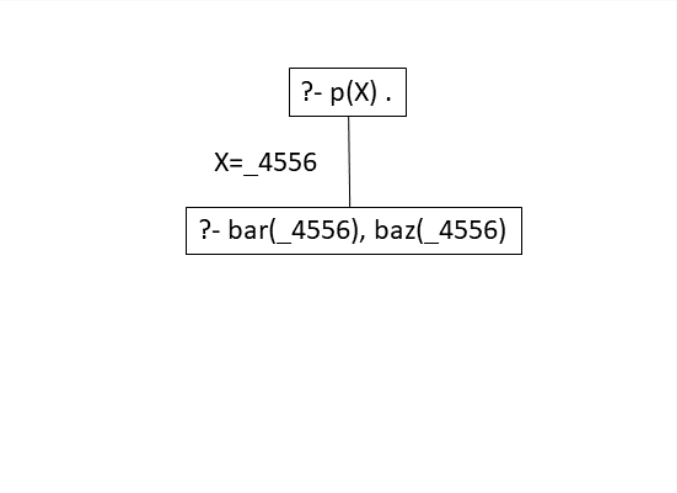
Să presupunem că avem programul:

```
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.

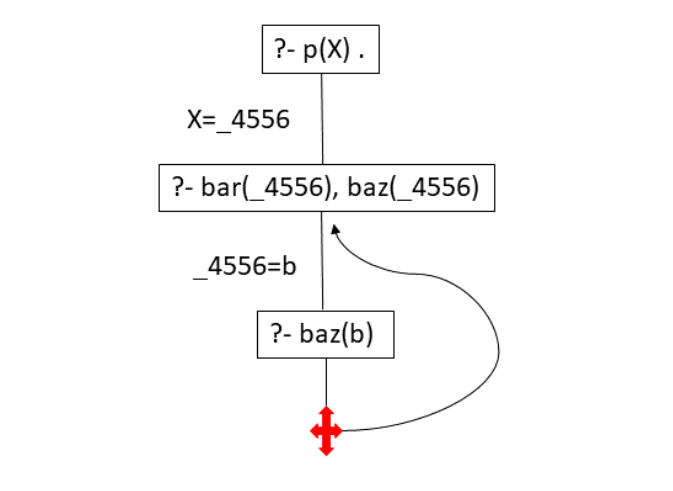
Exemplu

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.

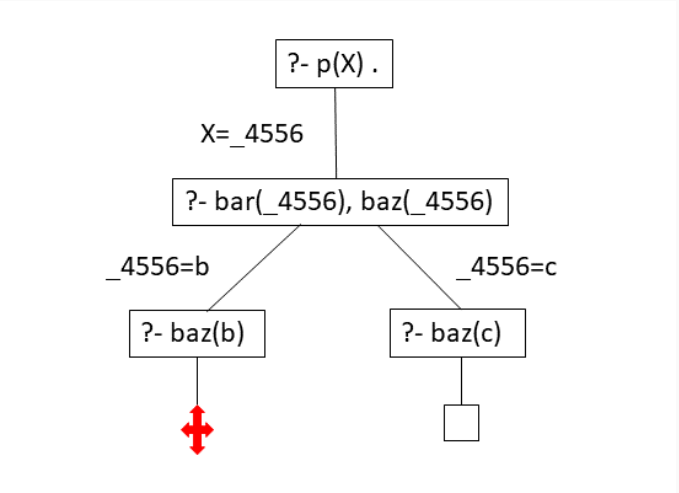
Exemplu

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
```

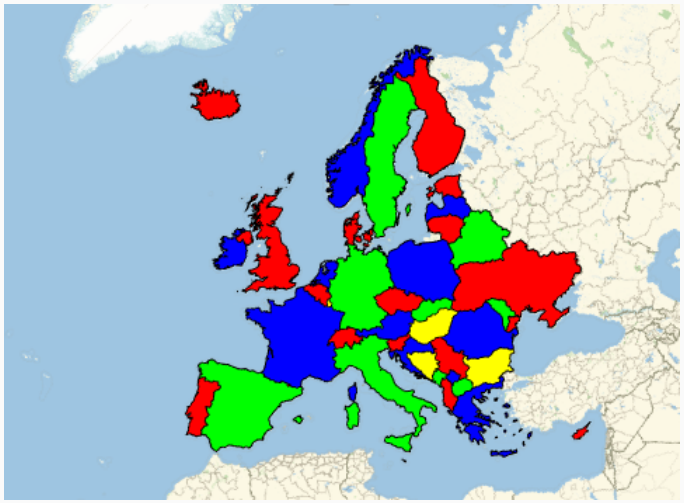
Vă explicați ce s-a întâmplat? Desenați arborele de căutare!

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

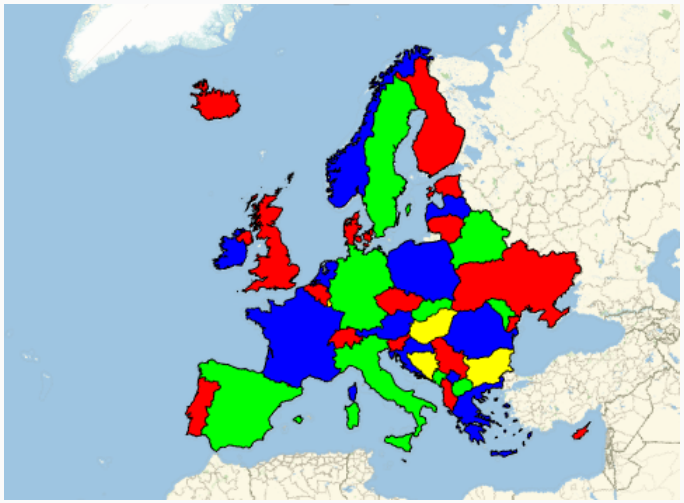
Un program mai complicat

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

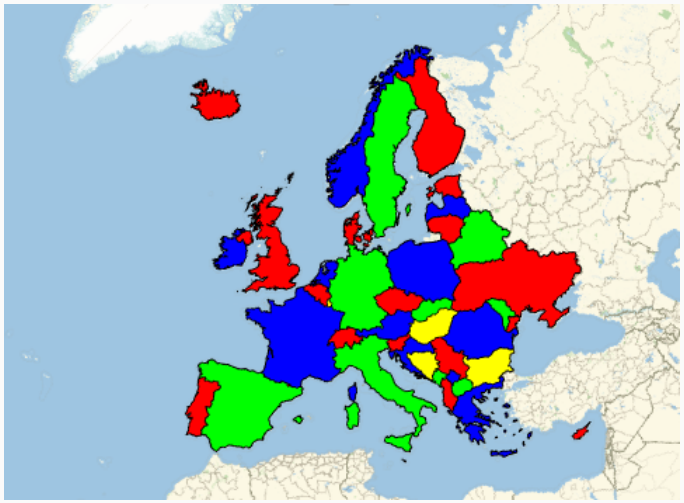
Să se coloreze o hartă dată cu un număr minim de culori astfel încât oricare două țări vecine să fie colorate diferit.

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 \= 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.
```

```
?- 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 forțează evaluarea (e.g., `=:=`).

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 \+

- 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 folosește 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')

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.

kb1: Un alt exemplu

Un program Prolog definește o bază de cunoștințe.

Exemplu

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

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

Acest program conține două predicate:

`bigger/2`, `is_bigger/2`.

Definirea predicatelor

- Predicate cu același nume, dar cu arități diferite, sunt predicate diferite.
- Scriem `foo/n` pentru a indica că un predicat `foo` are aritatea `n`.
- Predicatele pot avea aritatea 0 (nu au argumente); sunt predefinite în limbaj (`true, false`).
- Predicate predefinite: `X=Y` (este adevărat dacă X poate fi unificat cu Y); `X\=Y` (este adevărat dacă X nu poate fi unificat cu Y);

Un exemplu cu fapte și reguli

- O **regulă** este o afirmație de forma `Head :- Body.` unde
  - `Head` este un predicat (termen complex)
  - `Body` este o secvență de predicate, separate prin virgulă.

Exemplu

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

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

Exemplu

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

Reguli

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

Exemplu

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

Interpretarea:

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

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

Definirea predicatelor

Mai multe reguli care au același `Head` pot fi gândite ca având sau între ele.

Exemplu

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

Ele se pot uni (nu este totdeauna recomandat) folosind `;`.

Exemplu

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

Sintaxă: program

Un program în Prolog este o colecție de fapte și reguli.

Faptele și regulile trebuie grupate după atomii folosiți în Head.

Exemplu

Corect:

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

Incorect:

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

Întrebări, răspunsuri, ținte

- O `întrebare` (*query*) este o secvență de forma

  ```
  ?- p1(t1,...,tn),...,pn(t1',...,tn').
  ```

- Fiind dată o întrebare (deci o țintă), Prolog caută răspunsuri.
  - `true`/ `false` dacă întrebarea nu conține variabile;
  - dacă întrebarea conține variabile, atunci sunt căutate valori care fac toate predicatele din întrebare să fie satisfăcute; dacă nu se găsesc astfel de valori, răspunsul este `false`.

- Predicatele care trebuie satisfăcute pentru a răspunde la o întrebare se numesc ținte (goals).

Exemple de întrebări și răspunsuri

Exemplu

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

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

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

?- bigger(donkey, dog).
true

?- is_bigger(elephant, dog).
true

?- is_bigger(monkey, dog).
false

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

În varianta online, puteți adăuga întrebări la finalul programului ca în exemplul de mai jos. Întrebările vor apărea în lista din *Examples* (partea dreaptă).

Exemplu

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

/** <examples>

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

Un exemplu cu date și reguli ce conțin variabile

Exemplu

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

Diverse

- Un program în Prolog are extensia `.pl`
- Comentarii:

```
% comentează restul liniei

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

- Nu uitați să puneți . la sfârșitul unui fapt sau al unei reguli.
- un program (o bază de cunoștințe) se încarcă folosind:

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

Practică

Exercițiul 1
încercați să răspundeți la următoarele întrebări, verificând in 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.

Negația

În Prolog există predicatul predefinit `not` cu următoarea semnificație:

`not(goal)` este `true` dacă `goal` nu poate fi demonstrat în baza de date curentă.

Atenție: `not` nu este o negație logică, ci exprimă imposibilitatea de a face demonstrația (sau instanțierea) conform cunoștințelor din bază ('closed world assumption'). Pentru a marca această distincție, în variantele noi ale limbajului, în loc de `not` se poate folosi operatorul `\+`.

Exemplu

```
not_parent(X,Y) :- not(parent_of(X,Y)). % sau
not_parent(X,Y) :- \+ parent_of(X,Y).
```

Exercițiul 3: negația
Folosind baza anterioară (arbore genealogic) testați predicatul `not_parent`:

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

Ce observați? Încercați să analizați răspunsurile primite.

Exercițiul 3: negația
Folosind baza anterioară (arbore genealogic) testați predicatul `not_parent`:

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

Ce observați? Încercați să analizați răspunsurile primite.

- Corectați `not_parent` astfel încât să dea răspunsul corect la toate întrebările de mai sus.

Bibliografie

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

Pe data viitoare!