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,

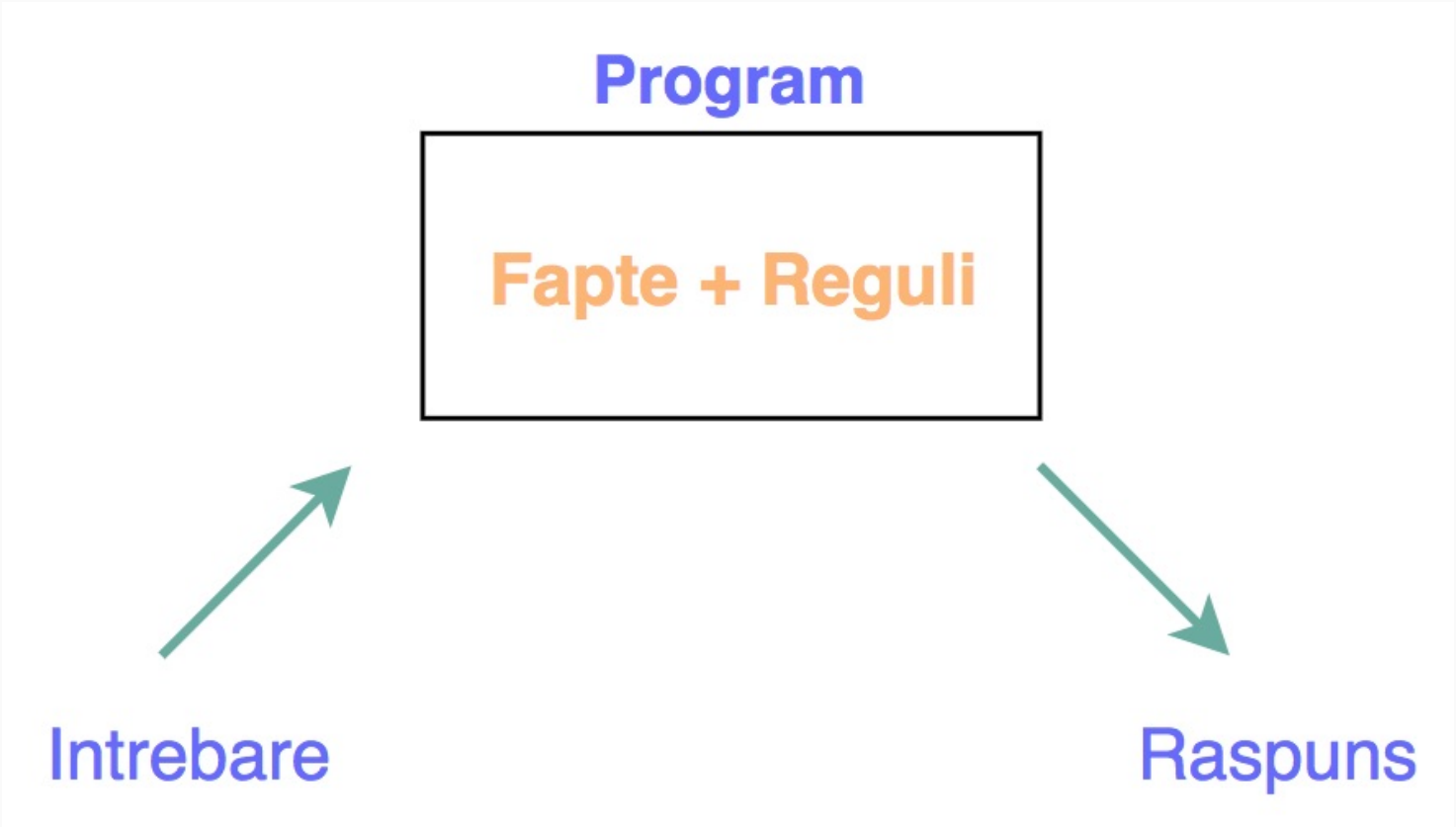
atunci

`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.

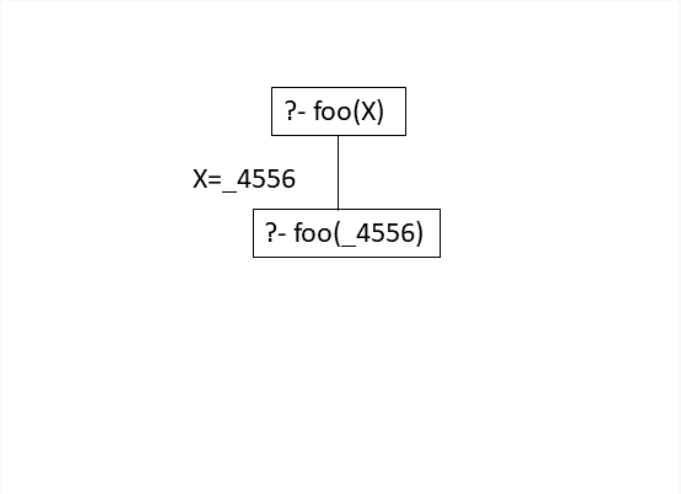
Exemplu

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.

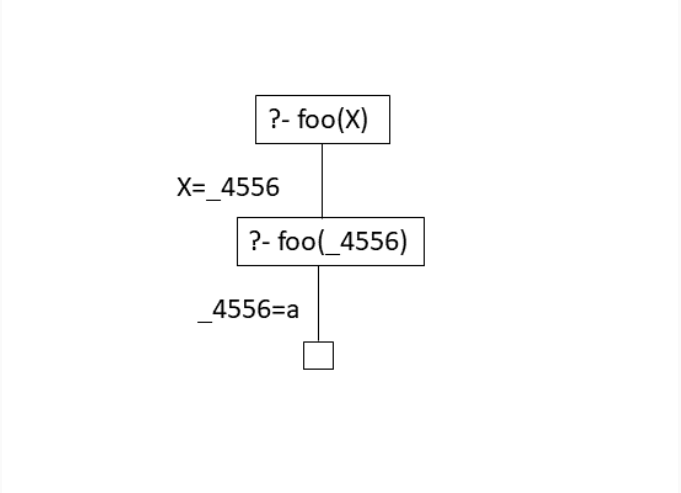
Exemplu

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.

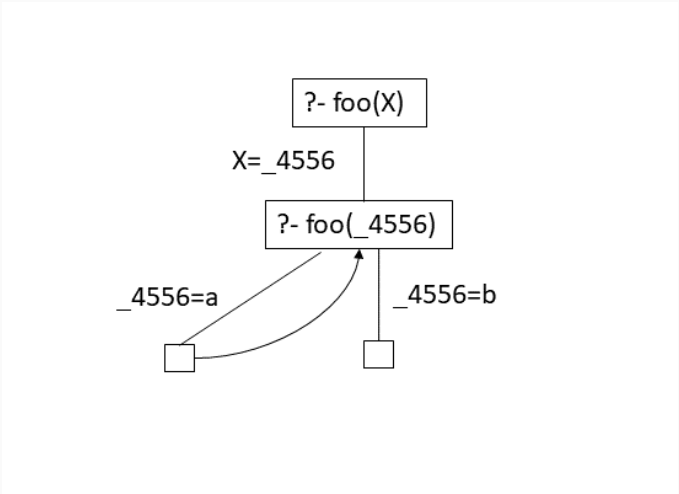
Exemplu

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.

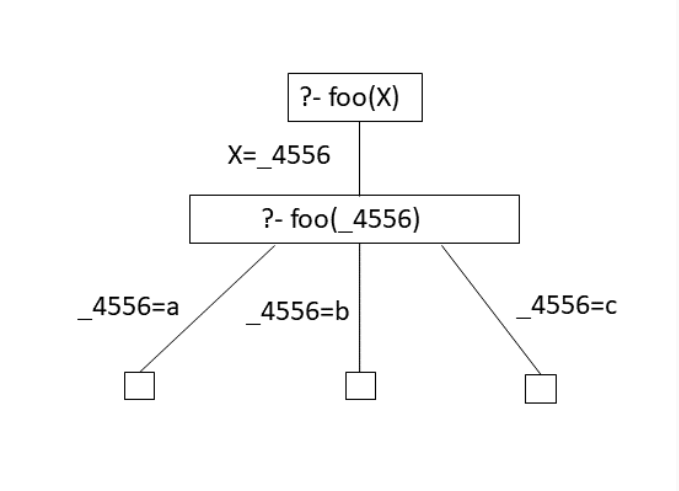
Cum găsește Prolog răspunsul

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

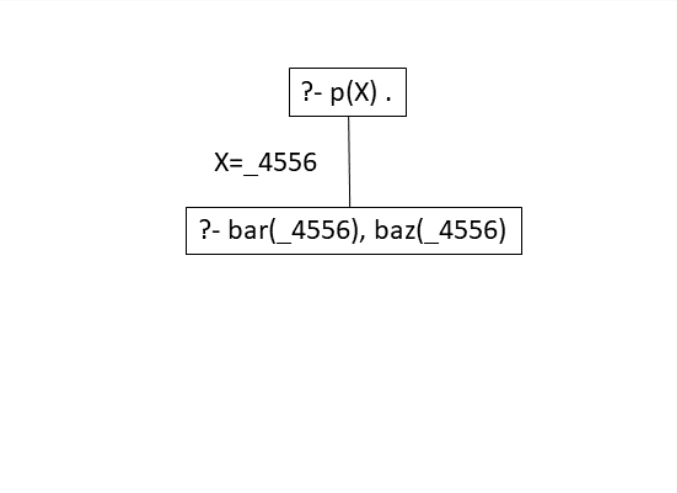
Să presupunem că avem programul:

```
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.

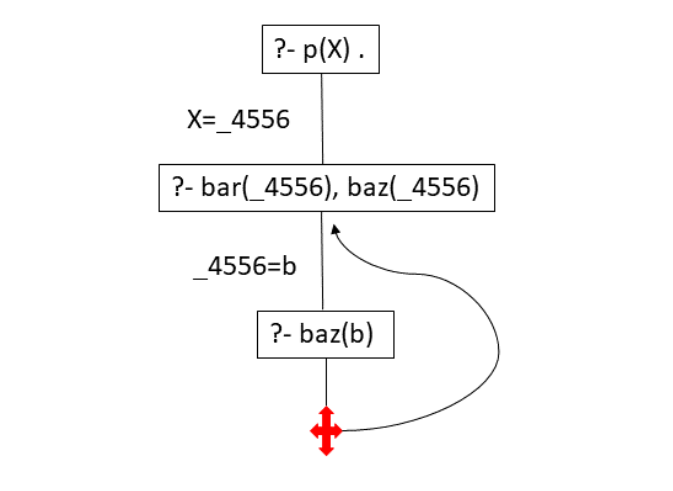
Exemplu

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.

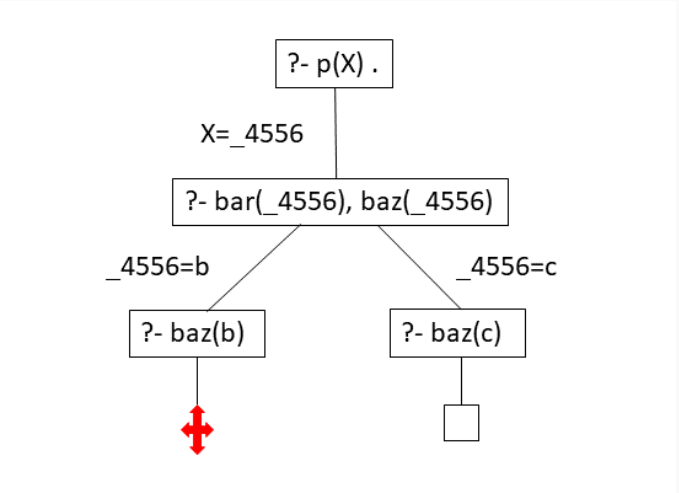
Exemplu

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
```

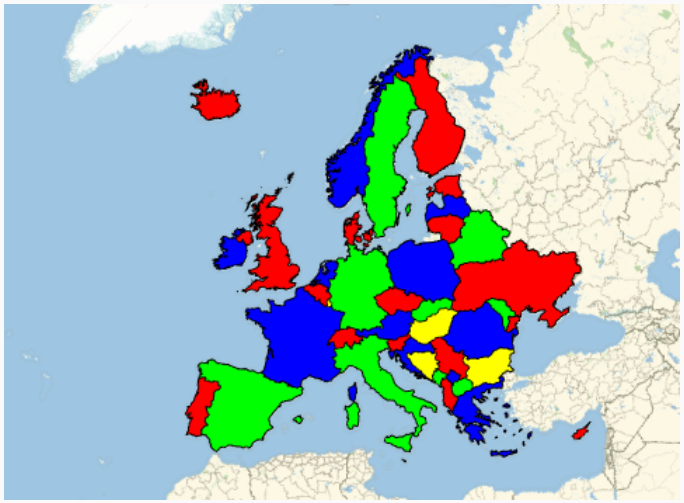
Vă explicați ce s-a întâmplat? Desenați arborele de căutare!

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

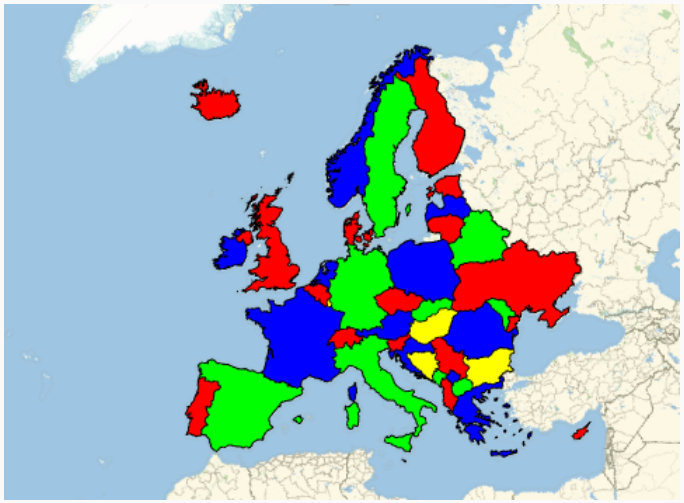
Un program mai complicat

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

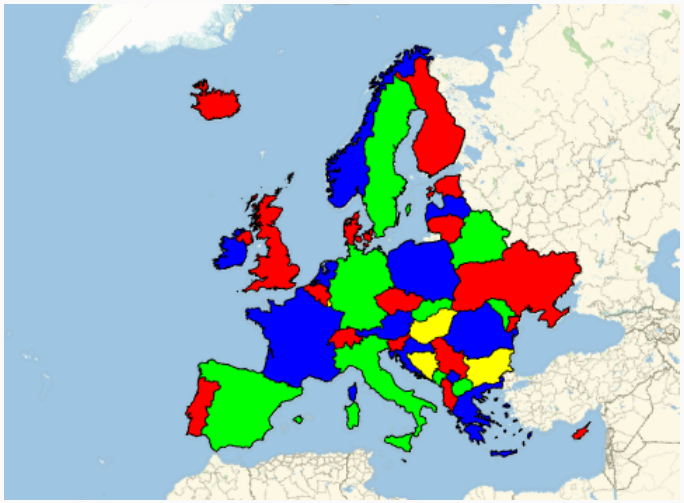
Să se coloreze o hartă dată cu un număr minim de culori astfel încât oricare două țări vecine să fie colorate diferit.

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!

Laborator 2

Logică matematică și computațională

Laboratorul 2

Cuprins

- Aritmetica în Prolog
- Recursivitate în Prolog
- Liste în Prolog

Material suplimentar

- `Capitolul 2` - `Capitolul 6` din *Learn Prolog Now!*.

Aritmetica în Prolog

Aritmetica în Prolog

Exemplu

```
?- 3+5 = +(3,5).
true
?- 3+5 = +(5,3).
false
?- 3+5 = 8.
false
```

Explicații:

- 3+5 este un termen.
- Prolog trebuie anunțat explicit pentru a îl evalua ca o expresie aritmetică, folosind predicate predefinite în Prolog, cum sunt `is/2`, `=:=/2`, `>/2` etc.

Aritmetica în Prolog

Exercițiu. *Analizați următoarele exemple:*

```
?- 3+5 is 8.
false

?= X is 3+5.
X = 8

?- 8 is 3+X.
is/2: Arguments are not sufficiently instantiated

?- X=4, 8 is 3+X.
false
```

Aritmetica în Prolog

Exercițiu. Analizați următoarele exemple:

```
?- X is 30-4.
X = 26

?- X is 3*5.
X = 15

?- X is 9/4.
X = 2.25
```

Aritmetica în Prolog

Operatorul `is`:

- Primește două argumente
- Al doilea argument trebuie să fie o expresie aritmetică validă, cu toate variabilele inițializate
- Primul argument este fie un număr, fie o variabilă
- Dacă primul argument este un număr, atunci rezultatul este `true` dacă este egal cu evaluarea expresiei aritmetice din al doilea argument.
- Dacă primul argument este o variabilă, răspunsul este pozitiv dacă variabila poate fi unificată cu evaluarea expresiei aritmetice din al doilea argument.

Totuși, nu este recomandat să folosiți `is` pentru a compara două expresii aritmetice, ci operatorul `=:=`.

Aritmetica în Prolog

Exercițiu. *Analizați următoarele exemple:*

```
?- 8 > 3.
true

?- 8+2 > 9-2.
true

?- 8 < 3.
false
```

```
?- 8 >= 3.
true

?- 8 =:= 3.
false

?- 8 =\= 3.
true
```

Operatori aritmetici

Operatorii aritmetici predefiniți în Prolog sunt de două tipuri:

- funcții
- relații (predicate)

Funcții

- Adunarea și înmulțirea sunt exemple de funcții aritmetice.
- Aceste funcții sunt scrise în mod uzual și în Prolog.

Exemplu

```
2 + (-3.2 * X - max(17,X)) / 2 ** 5
```

- 2**5 înseamnă 2^5
- Exemple de alte funcții disponibile:
  `min/2`, `abs/1` (modul), `sqrt/1` (radical), `sin/1` (sinus)
- Operatorul `//` este folosit pentru împărțire întreagă.
- Operatorul `mod` este folosit pentru restul împărțirii întregi.

Relații

- Relațiile aritmetice sunt folosite pentru a compara evaluarea expresiilor aritmetice (e.g, `X > Y`)
- Exemple de relații disponibile:
  <, >, =<, >=, =\= (diferit), =:= (aritmetic egal)
- Atenție la diferența dintre =:= și =:
  - =:= compară două expresii aritmetice
  - = caută un unificator

Exemplu

```
?- 2 ** 3 =:= 3 + 5.
true
?- 2 ** 3 = 3 + 5.
false
```

Exercițiul 1: distanța dintre două puncte
Definiți un predicat `distance/3` pentru a calcula distanța dintre două puncte într-un plan 2-dimensional. Punctele sunt date ca perechi de coordonate.

Exemple:

```
?- distance((0,0), (3,4), X).
X = 5.0

?- distance((-2.5,1), (3.5,-4), X).
X = 7.810249675906654
```

Recursivitate

Bază de cunoștințe

În laboratorul trecut, am folosit următoarea bază de cunoștințe:

```
parent(bob, lisa).
parent(bob, paul).
parent(bob, mary).
parent(juliet, lisa).
parent(juliet, paul).
parent(juliet, mary).

parent(peter, harry).
parent(lisa, harry).
parent(mary, dony).
parent(mary, sandra).
```

Recursivitate - strămoși

Am definit un predicat `ancestor_of(X,Y)` care este adevărat dacă `X` este un strămoș al lui `Y`.

Definiția recursivă a predicatului `ancestor_of(X,Y)` :

```
ancestor_of(X,Y) :- parent(X,Y).
ancestor_of(X,Y) :- parent(X,Z), ancestor_of(Z,Y).
```

Exercițiul 2: numerele Fibonacci
Scrieți un predicat `fib/2` pentru a calcula, pentru orice n, numărul de pe poziția n din șirul Fibonacci. Secvența de numere Fibonacci este definită prin: $F_0 := 1$, $F_1 := 1$, iar pentru orice $n \geq 2$,

$$F_n := F_{n-1} + F_{n-2}.$$

Exemple:

```
?- fib(1,X).
X=1.
true
```

```
?- fib(5,X).
X=8.
true
```

```
?- fib(2,X).
X=2.
true
```

Exercițiul 2 (cont.)
Programul scris anterior vă găsește răspunsul la întrebarea de mai jos?

```
?- fib(50,X).
```

Dacă da, felicitări! Dacă nu, încercați să găsiți o soluție mai eficientă!

Afișări în Prolog

- Pentru afișare se folosește predicatul `write/1`.
- Predicatul `nl/0` conduce la afișarea unei linii goale.

Exemplu

```
?- write('Hello World!'), nl.
Hello World!
true

?- X = hello, write(X), nl.
hello
X = hello
```

Exercițiul 3: afișarea unui pătrat de caractere
Scrieți un program în Prolog pentru a afișa un pătrat de $n \times n$ caractere pe ecran.

Denumiți predicatul `square/2`. Primul argument este un număr natural diferit de 0, iar al doilea un caracter care trebuie afișat.

Exemplu:

```
?- square(5, '* ').
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
```

Liste

Liste

- Listele în Prolog sunt un tip special de date (termeni speciali).
- Listele se scriu între paranteze drepte, cu elementele despărțite prin virgulă.
- `[]` este lista vidă.

Exemplu

- `[elephant, horse, donkey, dog]`
- `[elephant, [], X, parent(X, tom), [a, b, c], f(22)]`

Head & Tail

- Primul element al unei liste se numește *head*, iar restul listei *tail*.
- Evident, o listă vidă nu are un prim element.
- În Prolog există o notație utilă pentru liste cu separatorul `|`, evidențiind primul element și restul listei.

Exemplu

```
?- [1, 2, 3, 4, 5] = [Head | Tail].
Head = 1
Tail = [2, 3, 4, 5]
```

Cu această notație putem să returnăm ușor, de exemplu, al doilea element dintr-o listă.

```
?- [quod, licet, jovi, non, licet, bovi] = [_, X | _].
X = licet
```

Exemplu (`element_of/2`)

- un predicat care verifică dacă o listă conține un anumit termen
- `element_of(X,Y)` trebuie să fie adevărat dacă `X` este un element al lui `Y`.

```
/* Dacă primul element al listei este termenul
pe care îl căutăm, atunci am terminat. */

element_of(X,[X|_]).

% Altfel, verificăm dacă termenul se află în restul
listei.

element_of(X,[_|Tail]) :- element_of(X,Tail).

?- element_of(a,[a,b,c]).
?- element_of(X,[a,b,c]).
```

Exemplu (`concat_lists/3`)

- un predicat care este poate fi folosit pentru a concatena două liste
- al treilea argument este concatenarea listelor date ca prime două argumente

```
concat_lists([], List, List).
concat_lists([Elem | List1], List2, [Elem | List3]) :-
      concat_lists(List1, List2, List3).
```

```
?- concat_lists([1, 2, 3], [d, e, f, g], X).
?- concat_lists(X, Y, [a, b, c, d]).
```

Lucrul cu liste

În Prolog există niște predicate predefinite pentru lucrul cu liste. De exemplu:

- `length/2`: al doilea argument întoarce lungimea listei date ca prim argument
- `member/2`: este adevărat dacă primul argument se află în lista dată ca al doilea argument
- `append/3`: identic cu predicatul anterior `concat_lists/3`
- `last/2`: este adevărat dacă al doilea argument este identic cu ultimul element al listei date ca prim argument
- `reverse/2`: lista din al doilea argument este lista dată ca prim element în oglindă.

Exercițiul 4

A) Definiți un predicat `all_a/1` care primește ca argument o listă și care verifică dacă argumentul său este format doar din a-uri.

```
?- all_a([a,a,a,a]).
?- all_a([a,a,A,a]).
```

B) Scrieti un predicat `trans_a_b/2` care traduce o listă de a-uri într-o listă de b-uri. `trans_a_b(X,Y)` trebuie să fie adevărat dacă `X` este o listă de a-uri și `Y` este o listă de b-uri, iar cele două liste au lungimi egale.

```
?- trans_a_b([a,a,a],L).
?- trans_a_b([a,a,a],[b]).
?- trans_a_b(L,[b,b]).
```

Exercițiul 5: Operații cu vectori
A) Scrieți un predicat `scalarMult/3`, al cărui prim argument este un întreg, al doilea argument este o listă de întregi, iar al treilea argument este rezultatul înmulțirii cu scalari al celui de-al doilea argument cu primul.

De exemplu, la întrebarea

```
?-scalarMult(3,[2,7,4],Result).
```

ar trebui să obțineți `Result = [6,21,12]`.

Exercițiul 5 (cont.)
B) Scrieți un predicat `dot/3` al cărui prim argument este o listă de întregi, al doilea argument este o listă de întregi de lungimea primeia, iar al treilea argument este produsul scalar dintre primele două argumente.

De exemplu, la întrebarea

```
?-dot([2,5,6],[3,4,1],Result).
```

ar trebui să obțineți `Result = 32`.

Exercițiul 5 (cont.)
C) Scrieți un predicat `max/2` care caută elementul maxim într-o listă de numere naturale.

De exemplu, la întrebarea

```
?-max([4,2,6,8,1],Result).
```

ar trebui să obțineți `Result = 8.`

Laborator 3

Logică matematică și computațională

Laboratorul 3

Cuprins

- Alte exerciții cu liste
- Sortări

Alte exerciții cu liste

Practică

Exercițiul 1
Definiți un predicat `palindrome/1` care este adevărat dacă lista primită ca argument este palindrom (lista citită de la stânga la dreapta este identică cu lista citită de la dreapta la stânga).

De exemplu, la întrebarea

```
?- palindrome([r,e,d,i,v,i,d,e,r]).
```

ar trebui să obțineți `true`.

Nu folosiți predicatul predefinit `reverse`, ci propria implementare a acestui predicat.

Practică

Exercițiul 2
Definiți un predicat `remove_duplicates/2` care șterge toate duplicatele din lista dată ca prim argument și întoarce rezultatul în al doilea argument.

De exemplu, la întrebarea

```
?- remove_duplicates([a, b, a, c, d, d], List).
```

ar trebui să obțineți `List = [b, a, c, d]`.

Exercițiul 3
Definiți un predicat `atimes/3` care să fie adevărat exact atunci când elementul din primul argument apare în lista din al doilea argument de numărul de ori precizat în al treilea argument.

Interogați:

```
?- atimes(3,[3,1,2,1],X).
?- atimes(1,[3,1,2,1],X).
?- atimes(N,[3,1,2,1],2).
?- atimes(N,[3,1,2,1],1).
?- atimes(N,[3,1,2,1],0).
?- atimes(N,[3,1,2,1],X).
```

Sortări

Sortarea prin inserție (*insertion sort*)

Predicatul `insertsort/2` sortează lista de pe primul argument folosind algoritmul *insertion sort*.

```
insertsort([],[]).
insertsort([H|T],L) :- insertsort(T,L1), insert(H,L1,L).
```

Exercițiul 4: scrieți regulile care definesc comportamentul predicatului ajutător `insert/3`.

Quicksort

Predicatul `quicksort/2` sortează lista de pe primul argument folosind algoritmul *quicksort*.

```
quicksort([],[]).
quicksort([H|T],L) :-
split(H,T,A,B), quicksort(A,M), quicksort(B,N),
append(M,[H|N],L).
```

Exercițiul 5: scrieți regulile care definesc comportamentul predicatului ajutător `split/4`.

Laborator 4

Logică matematică și computațională

Găsirea soluțiilor

Introducere

Până acum, am implementat, în mare parte, predicate care reprezentau funcții, în sensul că intrarea funcției modelate de un asemenea predicat era reprezentată de primul argument (sau de primele argumente), iar ieșirea de ultimul argument (sau de ultimele argumente).

Acest mod de a programa era, însă, caracteristic programării funcționale, despre care se va vorbi la cursul omonim din anul II.

Forța Prolog-ului (și, în general, a programării logice) stă, de fapt, în lucrul cu predicate care nu reprezintă neapărat funcții, după cum se va vedea în acest laborator.

Exercițiul 1
Definiți un predicat `listaNelem/3` astfel încât, pentru orice `L`, `N`, `M`, `listaNelem(L,N,M)` este adevărat exact atunci când `M` este o listă cu `N` elemente care sunt toate elemente ale lui `L` (cu eventuale repetiții).

Interogați:

```
?- listaNelem([1,2,3],2,X).
?- listaNelem(L,1,[2]).
?- listaNelem(L,2,[2]).
?- listaNelem(L,2,[2,3]).
?- listaNelem(L,2,[2,3]), length(L,3).
?- length(L,3), listaNelem(L,2,[2,3]).
```

Toate soluțiile

Ce facem, însă, dacă vrem un predicat `listeNelem/3` astfel încât, pentru orice `L`, `N`, `LL`, `listeNelem(L,N,LL)` este adevărat exact atunci când `LL` este lista tuturor acelor `M` cu proprietatea că `listaNelem(L,N,M)`?

Există o soluție mai complexă care folosește doar conceptele introduse până acum, dar Prolog-ul ne furnizează și varianta:
`listeNelem(L,N,LL) :- bagof(M, listaNelem(L,N,M), LL).`

Pe lângă `bagof/3`, există mai multe asemenea „metapredicate”. Le vom descrie separat.

bagof/3

Interogați:

```
?- bagof((X,Y),
(member(X,[1,2,2,2,3]),member(Y,[0,1,2,3,4,5]),X<Y),L).
?- bagof(X,
(member(X,[1,2,2,2,3]),member(Y,[0,1,2,3,4,5]),X<Y),L).
?- bagof(X,
Y^(member(X,[1,2,2,2,3]),member(Y,[0,1,2,3,4,5]),X<Y),L).
```

În al doilea exemplu, variabila `Y` este *liberă*, așadar se caută soluții atât pentru ea, cât și pentru `L` (simultan). În al treilea exemplu, variabila `Y` este *cuantificată existențial*, în sensul că se caută „toți `X` astfel încât există `Y` astfel încât...".

setof/3

Interogați:

```
?- setof((X,Y),
(member(X,[1,2,2,2,3]),member(Y,[0,1,2,3,4,5]),X<Y),L).
?- setof(X,
(member(X,[1,2,2,2,3]),member(Y,[0,1,2,3,4,5]),X<Y),L).
?- setof(X,
Y^(member(X,[1,2,2,2,3]),member(Y,[0,1,2,3,4,5]),X<Y),L).
```

Comportamentul lui `setof/3` este similar cu cel al lui `bagof/3`, cu deosebirea că se încearcă eliminarea duplicatelor.

findall/3

Interogați:

```
?- findall(X,
(member(X,[1,2,2,2,3]),member(Y,[0,1,2,3,4,5]),X<Y),L).
?- findall((X,Y),
(member(X,[1,2,2,2,3]),member(Y,[0,1,2,3,4,5]),X<Y),L).
```

Comportamentul lui `findall/3` este similar cu cel al lui `bagof/3`, cu două deosebiri. În primul rând, semnul de cuantificare existențială nu mai este permis și, în același timp (sau chiar de aceea), orice variabilă așa-zis liberă va fi implicit cuantificată existențial.

În al doilea rând, cele două metapredicate au un comportament diferit atunci când nu există soluții:

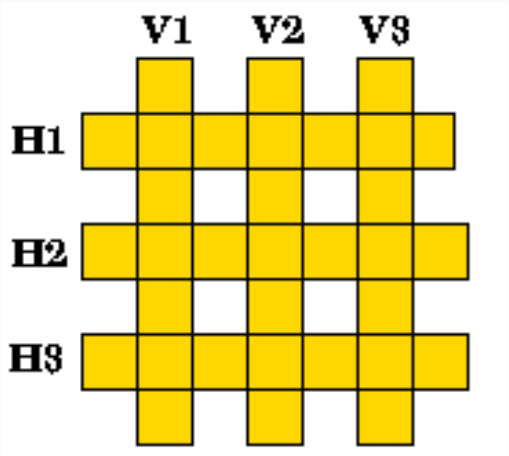
```
?- bagof(X, (member(X,[1,2,2,2,3]),0 is 1),L).
?- findall(X, (member(X,[1,2,2,2,3]),0 is 1),L).
```

Puzzle-uri simple

Exercițiul 2: cuvinte încrucișate
Șase cuvinte din engleză, anume:

abalone, abandon, anagram, connect, elegant, enhance

trebuie aranjate într-un puzzle de cuvinte încrucișate, ca în figură.



Exercițiul 2 (cont.)
Pornind de la faptele

```
word(abalone,a,b,a,l,o,n,e).
word(abandon,a,b,a,n,d,o,n).
word(anagram,a,n,a,g,r,a,m).
word(connect,c,o,n,n,e,c,t).
word(elegant,e,l,e,g,a,n,t).
word(enhance,e,n,h,a,n,c,e).
```

definiți un predicat `crosswd/6` care calculează toate variantele în care puteți completa grila. Primele trei argumente trebuie să fie cuvintele pe verticală, de la stânga la dreapta (V1, V2, V3), iar următoarele trei argumente trebuie să fie cuvintele pe orizontală, de sus în jos (H1, H2, H3).

Hint: Specificați că V1, V2, V3, H1, H2, H3 sunt cuvinte care au anumite litere comune. Unde este cazul, folosiți variabile anonime.

Exercițiul 3: labirint

Următoarele fapte (care se continuă pe slide-ul următor) descriu un labirint:

```
connected(1,2).
connected(3,4).
connected(5,6).
connected(7,8).
connected(9,10).
connected(12,13).
connected(13,14).
connected(15,16).
connected(17,18).
connected(19,20).
```

Exercițiul 3 (cont.)

```
connected(4,1).
connected(6,3).
connected(4,7).
connected(6,11).
connected(14,9).
connected(11,15).
connected(16,12).
connected(14,17).
connected(16,19).
```

Faptele indică ce puncte sunt conectate (din ce punct se poate ajunge într-un alt punct într-un pas).

Drumurile sunt cu sens unic (se poate merge pe ele doar într-o direcție).

De exemplu, se poate ajunge într-un pas de la 1 la 2, dar nu și invers.

Exercițiul 3 (cont.)

Adăugați un predicat `path/3` care indică dacă dintr-un punct se poate ajunge într-un alt punct, în mai mulți pași, cel de-al treilea argument reprezentând lista pașilor. Pe baza lui, construiți un predicat `pathc/2` care spune doar dacă dintr-un punct se poate ajunge într-un alt punct.

Interogați:

```
?- pathc(5,10).
?- path(5,10,L).
?- pathc(6,X).
?- path(6,X,L).
?- pathc(X,13).
```

Lucrul cu Prolog-ul pe desktop

Utilizarea SWI-Prolog

Pe lângă SWISH, putem lucra și în aplicația desktop SWI-Prolog. Programul propriu-zis se scrie într-un fișier, care se încarcă din consola programului, unde se vor scrie și interogările.

Comenzi utile:

- `pwd.` pentru a afla directorul curent;
- `[fisier].` pentru încărcarea fișierului `fisier.pl` (el trebuie reîncărcat la fiecare modificare);
- `;` pentru a se genera următoarea soluție;
- *enter* pentru a nu mai căuta altă soluție;
- CTRL+C și a pentru oprirea unei căutări în desfășurare.

Puzzle-ul *Countdown*

Exercițiul 4: cel mai lung cuvânt
Acest exemplu provine din
Ulle Endriss, *Lecture Notes – An Introduction to Prolog Programming.*
și a mai fost folosit în trecut în laboratoare de programare logică în FMI.

Countdown este un joc de televiziune popular în Marea Britanie în care jucătorii trebuie să găsească un cuvânt cât mai lung cu literele dintr-o mulțime dată de nouă litere.

Să încercăm să rezolvăm acest joc cu Prolog!

Exercițiul 4 (cont.)
Concret, vom încerca să găsim o soluție optimă pentru următorul joc:

Primind o listă cu litere din alfabet (nu neapărat unice),
trebuie să construim cel mai lung cuvânt format din literele date
(pot rămâne litere nefolosite).

Vom rezolva jocul pentru cuvinte din limba engleză.

Scorul obținut este lungimea cuvântului găsit.

Exercițiul 4 (cont.)
Scopul final este de a construi un predicat în Prolog `topsolution/2`, cu următorul comportament:

dându-se o listă de litere în primul său argument, trebuie să returneze în al doilea argument o soluție cât mai bună, adică un cuvânt din limba engleză de lungime maximă care poate fi format cu literele din primul argument.

```
?- topsolution([r,d,i,o,m,t,a,p,v],Word).
Word = dioptra
```

Exercițiul 4 (cont.)
Începeți prin a descărca fișierul `words.pl` în același director cu fișierul programului vostru.

Acest fișier conține o listă cu peste 350.000 de cuvinte din limba engleză, de la `a` la `zyzzyva`, sub formă de fapte.

Scrieți `[words].` în consolă pentru a încărca aceste fapte.

Exercițiul 4 (cont.)
Predicatul predefinit din Prolog `atom_chars(Atom,CharList)` descompune un atom într-o listă de caractere.

Folosiți acest predicat pentru a defini un predicat `word_letters/2` care transformă un cuvânt (i.e, un atom în Prolog) într-o listă de litere.
De exemplu:

```
?- word_letters(hello,X).
X = [h,e,l,l,o]
```

Ca o paranteză, observați că puteți folosi acest predicat pentru a găsi cuvinte în engleză de 45 de litere:

```
?- word(Word), word_letters(Word,Letters),
   length(Letters,45).
```

Exercițiul 4 (cont.)
Mai departe, scrieți un predicat `cover/2` care, primind două liste, verifică dacă a doua listă „acoperă” prima listă (i.e., verifică dacă fiecare element care apare de k ori în prima listă apare de cel puțin k ori în a doua listă).

De exemplu

```
?- cover([a,e,i,o], [m,o,n,k,e,y,b,r,a,i,n]).
true

?- cover([e,e,l], [h,e,l,l,o]).
false
```

Exercițiul 4 (cont.)
Scrieți un predicat `solution/3` care primind o listă de litere ca prim argument și un scor dorit ca al treilea argument, returnează prin al doilea argument un cuvânt cu lungimea egală cu scorul dorit, „acoperit” de lista respectivă de litere.

De exemplu

```
?- solution([g,i,g,c,n,o,a,s,t], Word, 3).
Word = act
```

Exercițiul 4 (cont.)
Implementați acum predicatul `topsolution/3`.

Testați, de exemplu, predicatul definit pe mulțimea de litere:

```
[y,c,a,l,b,e,o,s,x]
```

Aceasta este una dintre listele de litere folosite în ediția de *Countdown* din 18 decembrie 2002 din Marea Britanie, în care Julian Fell a obținut cel mai mare scor din istoria concursului. Pentru lista de mai sus, el a găsit cuvântul *cables*, câștigând astfel 6 puncte.

Poate programul vostru să bată acest scor?

Laborator 5

Logică matematică și computațională

Introducere

La acest laborator, vom implementa în Prolog formulele propoziționale și semantica lor.

Variabilele vor fi reprezentate de atomi Prolog, iar operatorii $\neg$, $\wedge$, $\vee$, $\rightarrow$ (pe care îi vom implementa individual, spre deosebire de curs/seminar) de simbolurile de funcție `non`, `si`, `sau`, `imp`.

Interogați:

```
?- X = a.
?- X = si(a,b).
?- X = imp(non(a),imp(a,b)).
```

Scopul laboratorului va fi determinarea algoritmică a faptului că o formulă este sau nu tautologie.

Practică

Exercițiul 1
Definiți un predicat `vars/2` care este adevărat exact atunci când primul argument este o formulă, iar al doilea argument este lista care reprezintă mulțimea variabilelor care apar în ea.

Exemplu:

```
?- vars(imp(non(a),imp(a,b)),S).
S = [a, b]
```

Indicii:

1. Folosiți predicatul predefinit `atom/1`.
2. Folosiți predicatul predefinit `union/3`, care calculează reuniunea a două liste considerate ca fiind mulțimi.

Exercițiul 2

În teoria mulțimilor, graficul unei funcții de la o mulțime A la o mulțime B este „implementat" ca o submulțime a lui $A \times B$. În acest fel vom implementa și evaluările propoziționale de forma $e : V \to \{0, 1\}$, unde V, spre deosebire de curs/seminar, va fi o mulțime (listă) finită de variabile.

De exemplu, o evaluare pe mulțimea de variabile $\{\mathtt{a}, \mathtt{b}\}$ poate fi `[(a,1),(b,0)]`.

Definiți un predicat `val/3`, astfel încât, pentru orice variabilă `V` și orice evaluare `E`, avem că, pentru orice `A`, `val(V,E,A)` este adevărat exact atunci când `A` este „`E(V)`".

Exemplu:

```
?- val(b,[(a,1),(b,0)],A).
A = 0
```

Exercițiul 3
Definiți predicate `bnon/2`, `bsi/3`, `bsau/3`, `bimp/3` care implementează operațiile $\neg$, $\wedge$, $\vee$, $\rightarrow$ pe mulțimea $\{0, 1\}$.

Exemple:

```
?- bsi(1,0,C).
C = 0
?- bimp(A,0,0).
A = 1
?- bimp(0,B,0).
false
```

Indiciu: Puteți defini unele operații în funcție de altele.

Exercițiul 4
Definiți un predicat `eval/3`, astfel încât, pentru orice formulă `X` și orice evaluare `E`, avem că, pentru orice `A`, `eval(X,E,A)` este adevărat exact atunci când `A` este „$\mathtt{E}^{+}\mathtt{(X)}$”.

Exemple:

```
?- eval(imp(b,d),[(a,1), (b,0), (d,1)],A).
A = 1
?- eval(imp(d,b),[(a,1), (b,0), (d,1)],A).
A = 0
```

Practică

Exercițiul 5
Definiți un predicat `evals/3`, astfel încât, pentru orice formulă `X` și orice listă de evaluări `Es`, avem că, pentru orice `As`, `evals(X,Es,As)` este adevărat exact atunci când `As` este lista rezultatelor evaluării lui `X` în fiecare dintre elementele lui `Es`.

Exemplu:

```
?- evals(imp(d,b),[[(a,1), (b,0), (d,1)], [(a,1), (b,1),
(d,0)]],As).
As = [0, 1]
```

Exercițiul 6
Definiți un predicat `evs/2`, astfel încât, pentru orice listă de variabile `S`, avem că, pentru orice `Es`, `evs(S,Es)` este adevărat exact atunci când `Es` este lista evaluărilor definite pe `S`.

Exemplu:

```
?- evs([c,b],Es).
Es = [[(c,0), (b,0)], [(c,1), (b,0)], [(c,0), (b,1)],
[(c,1), (b,1)]]
```

Indicii:

1. Pentru orice mulțime A există o unică funcție de la $\emptyset$ la A. De ce? Care este graficul ei?
2. Pentru pasul inductiv, definiți un predicat ajutător.

Practică

Exercițiul 7
Definiți un predicat `all_evals/2`, astfel încât, pentru orice formulă `X`, avem că, pentru orice `As`, `all_evals(X,As)` este adevărat exact atunci când `As` este lista rezultatelor evaluării lui `X` în fiecare dintre elementele listei evaluărilor definite pe *Var*(`X`).

Exemple:

```
?- all_evals(imp(a,a),As).
As = [1, 1]
?- all_evals(imp(a,b),As).
As = [1, 0, 1, 1]
```

Practică

Exercițiul 8
Definiți un predicat `taut/1`, astfel încât, pentru orice formulă `X`, avem că `taut(X)` este adevărat exact atunci când `X` este tautologie.

Exemple:

```
?- taut(imp(a,a)).
true
?- taut(imp(a,b)).
false
```

Laborator 6

Logică matematică și computațională

Inversarea listelor

Predicatul următor, `listN/2`, va fi util pentru generarea unei liste de lungime dată:

```
listN([],0).
listN([a|T], N) :- N > 0, M is N - 1, listN(T,M).
```

Introducem și metapredicatul `listing/1`, care afișează toate clauzele corespunzătoare unui predicat. Interogați:

```
?- listing(listN).
```

Reamintim, acum, din soluțiile Laboratorului 3, definirea predicatului `rev/2` de inversare a listelor:

```
rev([],[]).
rev([H|T],L) :- rev(T,N), append(N,[H],L).
```

Soluția dată nu este prea eficientă, având o complexitate pătratică.

Acumulatori

Soluția următoare o îmbunătățește pe cea precedentă, adăugând un predicat auxiliar, care are un parametru în plus, care joacă rol de acumulator. Complexitatea devine liniară (testați pentru liste de lungime 1000-10000):

```
reva(L,R) :- revah(L,[],R).
revah([], R, R).
revah([H|T], S, N) :- revah(T,[H|S],N).
```

Contemplați adevărul următoarei afirmații: pentru orice `A`, `B`, `C`, avem că `revah(A,B,C)` dacă și numai dacă, notând cu `M` inversa listei `A`, avem că `append(M,B,C)`.

În continuare, ținând cont de această afirmație, vom rescrie soluția de mai sus, permițând generalizarea ei la alte probleme.

Difference lists

Reamintim că afirmația era: pentru orice `A`, `B`, `C`, avem că `revah(A,B,C)` dacă și numai dacă, notând cu `M` inversa listei `A`, avem că `append(M,B,C)`. Altfel spus, inversa lui `A` este „`C` fără `B`".

Vom reprezenta expresia „`C` fără `B`" sub forma unei perechi `(C,B)` și o vom numi *difference list* sau **difflist**.

Definiția anterioară devine:

```
revd(L,R) :- revdh(L,(R,[])).
revdh([],(R,R)).
revdh([H|T],(N,S)) :- revdh(T,(N,[H|S])).
```

Practică

Exercițiul 1
Definiți un predicat `flatten/2` care aplatizează structura unei liste.

Exemplu:

```
?- flatten([1,2,[3,a],[[7],2],5],L).
L = [1, 2, 3, a, 7, 2, 5]
```

Dați o soluție care folosește `append/3` și una care folosește difflist-uri.

Indiciu: Folosiți metapredicatul `is_list/1`.

Exercițiul 2
Reamintim, tot din soluțiile Laboratorului 3, definirea predicatului `quicksort/2`:

```
quicksort([],[]).
quicksort([H|T],L) :- split(H,T,A,B), quicksort(A,M),
                      quicksort(B,N), append(M,[H|N],L).
split(_,[],[],[]).
split(X,[H|T],[H|A],B) :- H < X, split(X,T,A,B).
split(X,[H|T],A,[H|B]) :- H >= X, split(X,T,A,B).
```

Rescrieți această definiție folosind difflist-uri (fără a mai folosi `append/3`).