Unificarea si recursivitatea

Interogări

Introduceţi următoarele interogări si asiguraţi-vă ca înţelegeţi de ce unificarea în unele cazuri reuşeşte, iar în unele nu:

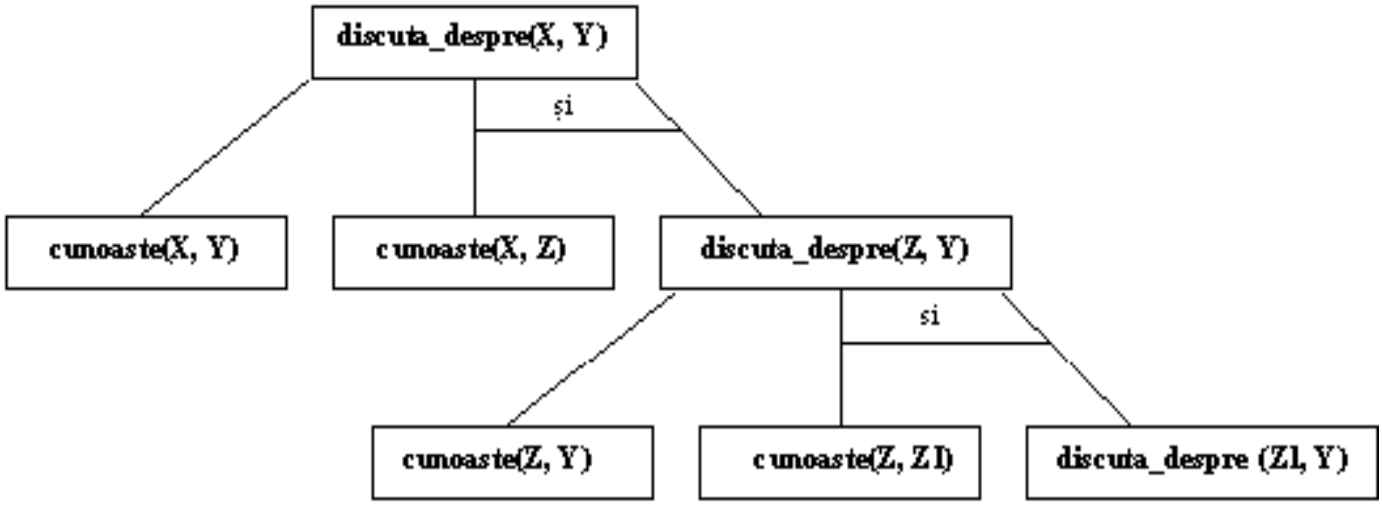
2 + 1 = 3.
f(X, a) = f(a, X).
marian = marian.
place(maria, X) = place(X, andrei).
f(X, Y) = f(P, P).

Recursivitatea

Vom ilustra recursivitatea prin următorul exemplu:

```
discuta_despre(A, B) :- cunoaste(A, B).
discuta_despre(X, Y) :- cunoaste(X, Z), discuta_despre(Z, Y).
```

Pentru a înţelege mai bine cum funcţionează recursivitatea putem sa ne imaginăm că pentru exemplul de mai sus, se formează următorul arbore care are clauze ca noduri:



Iată programul Prolog pe care îl puteţi testa:

```
cunoaste(maria, ana).
cunoaste(ana, mircea).
cunoaste(mircea, mihai).
discuta_despre(A, B) :- cunoaste(A, B).
discuta_despre(X, Y) :- cunoaste(X, Z), discuta_despre(Z, Y).
```

Folosim următoarele interogări:

cunoaste(X,_),
discuta_despre(X, Y), write(X), write(' discuta despre '), write(Y), nl, fail.

Exerciţii rezolvate

1. **Calculaţi factorialul unui număr.**

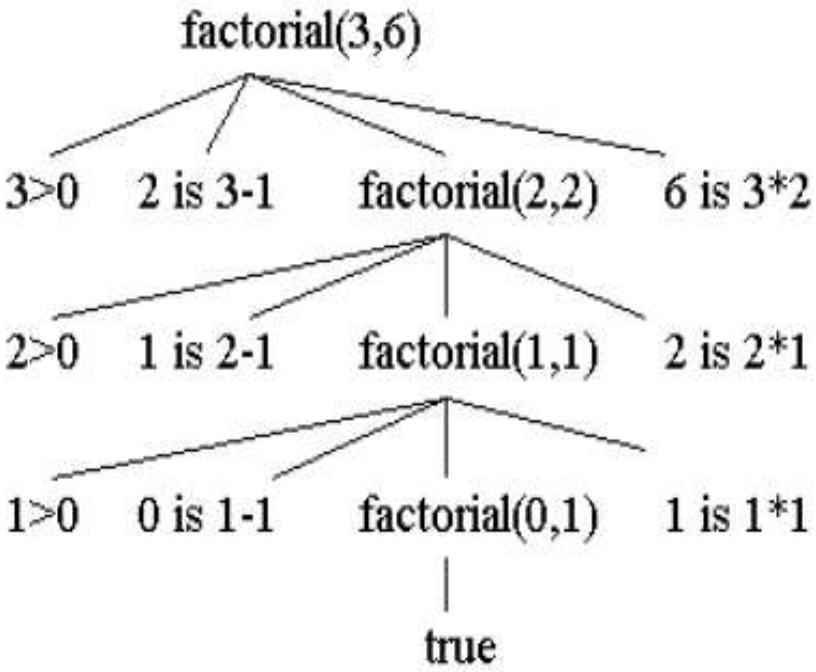
Vom rezolva aceasta problema folosind următoarea funcţie recursivă:

$$fact(x) = \begin{cases} 1, \text{ daca } x = 0; \\ fact(x - 1) * x, \text{ altfel} \end{cases}$$

```
factorial(0,1).
factorial(N,F) :- N>0, N1 is N-1, factorial(N1,F1), F is N * F1.
```

Pentru a calcula factorialul numărului 3, vom face o interogare după cum urmează:
factorial(3, F).
Răspuns: F = 6.

Următorul arbore este construit pentru interogarea *factorial(3, F)*. Ca noduri are clauze care nu conţin variabile libere, ci instanţe ale acestora:



O alta posibilitate de a realiza acest program este aceea de a folosi un predicat cu trei argumente astfel:

```
factorial(0,F,F).
factorial(N,A,F) :- N > 0, A1 is N*A, N1 is N -1, factorial(N1,A1,F).
```

Interogarea o vom realiza în felul următor:
factorial(3, 1, F).
Vom primi răspuns F = 6.

Calculul lui n-factorial n!=n*(n-1)!=n*(n-1)*(n-2)!=...= n*(n-1)*(n-2)*...*3*2*1
Ptr n=1 avem 1!=1
n=2 avem 2!=2*1!=2*1
n=3 avem 3!=3*2!=3*2*1!=3*2*1
n=4 avem 4!=4*3!=...=4*3*2*1 etc. ⇒

Funcţia recursivă va fi: **n!=1*2*3*...*(n-1)*n**

Algoritmul iterativ va fi:

Definiţia lui **n!=** **fact(n) = 1, dacă n=0**
n*fact(n-1), altfel

Mecanismul recursiv folosit este: n=3 **apel**
apel fact(3)=3*fact(2)=
=2*fact(1)=
=1*fact(0)=
=1

⇒ fact(0)=1, apoi
fact(1)=1*1,
fact(2)=1*1*2, apoi
fact(3)=3*2*1*1

main.pro – **v1 folosind *if***

```
% Factorial in stil procedural (similar C++)
implement main
    open core

class predicates
    factorial : (integer N, integer F) procedure (i,o).

clauses
    factorial(N,F) :-
        if N=0 then
            F=1
        else
            factorial(N-1,F1),
            F = N*F1
```

```
        end if.

clauses
    run():-
        console::init(),
        X=5,
        factorial(X,F),
        stdio::writef("%! = %",X,F),
        succeed().
end implement main

goal
    mainExe::run(main::run).
```

main.pro – **v2 folosind Prolog**

```
implement main
    open core

class predicates
    factorial : (integer N, integer F) procedure (i,o).

clauses

    factorial(0,F) :- F=1, !.

    factorial(N, F) :-
         factorial(N-1, F1),
         F = N * F1.

clauses
    run():-
        console::init(),
        X=5,
        factorial(X,F),
        stdio::writef("%! = %",X,F),
        succeed().
end implement main

goal
    mainExe::run(main::run).
```

Rulare

```
5! = 120
```

Rezolvaţi problema turnurilor din Hanoi.

Scopul acestui puzzle este de a muta n discuri de pe bara din stânga pe bara din dreapta folosind bara din centru ca pe una auxiliară. Important este că un disc mai mare nu poate fi aşezat pe un disc mai mic şi la un moment dat poate fi mutat numai unul. Imaginea următoare arată care este configuraţia de pornire pentru n = 3.

```
Muta discul din stanga in dreapta
Muta discul din stanga in centru
Muta discul din dreapta in centru
Muta discul din stanga in dreapta
Muta discul din centru in stanga
Muta discul din centru in dreapta
Muta discul din stanga in dreapta
Yes
```

```
% hanoi(1,X,Y,_):- write("Muta discul din ",X," in ",Y),nl.
% hanoi(N,X,Y,Z):- N>1, M is N-1, hanoi(M,X,Z,Y), hanoi(1,X,Y,Z),hanoi(M,Z,Y,X).
```

```
implement main
    open core

class predicates
    hanoi : (integer N, string X, string Y, string Z) procedure (i,i,i,i).

clauses
        hanoi(1,X,Y,Z) :-       !,
          stdio::writef("% -> %\n",X,Y).
        hanoi(N,X,Y,Z) :-
          M=N-1,
           hanoi(M,X,Z,Y),
           hanoi(1,X,Y,Z),
          hanoi(M,Z,Y,X).

clauses
    run():-
        console::init(),
```

```
        hanoi(3,"A","B","C"),
        succeed().
end implement main

goal
   mainExe::run(main::run).
```