

Paradigme de programare

Laborator 19

Mihai Nan

Facultatea de Automatica si Calculatoare
Universitatea Politehnica din Bucuresti

Anul universitar 2020-2021

1 Introducere

- Paradigma logic[Pleaseinsertintopreamble]
- Limbajul Prolog
- Sintaxă

2 Exemple

- Întrebări asupra unei baze de cunoștințe
- Liste
- Documentarea predicatelor și a argumentelor

3 Exerciții

- **Program** = colecție de reguli logice și fapte care reprezintă **baza de cunoștințe** a programatorului.
- **Rularea unui program** = găsirea de răspunsuri la întrebări puse de programator prin **raționament logic** bazat pe cunoștințele din program.
- **Programarea logică** este un stil de **programare declarativă**.
- **Faptele** sunt afirmații despre proprietățile obiectelor și relațiile existente între ele.
- O **regula**, în limbaj natural, este o frază de forma:
 - **Dacă** *ipoteza₁*, ... și *ipoteza_n*, atunci **concluzie**.

- Un program Prolog este o colecție de definiții ce descriu relații sau funcții de calculat - reprezentări simbolice de obiecte și relații între obiecte. Soluția problemelor nu se mai vede ca o execuție pas cu pas a unei secvențe de instrucțiuni.
- **Fapt** = ceea ce se cunoaște a fi adevărat.
- **Regulă** = ce se poate deduce din fapte date (indică o concluzie care se știe că e adevărată atunci când alte concluzii sau fapte sunt adevărate).
- **Concluzie** - Prolog folosește **rezoluția** pentru a demonstra dacă este adevărată sau nu, pornind de la ipoteza stabilită de faptele și regulile definite.

Constante

Numele constantelor încep cu litere mici.

Variabile

Variabila este un termen care începe cu o majusculă sau cu caracterul de subliniere (_). Variabilele sunt folosite pentru desemnarea obiectelor necunoscute. Există o **variabilă anonimă** (notată _) care este utilizată pentru a arăta ca pe poziția dată se acceptă orice valoare.

Fapte

Faptele sunt predicate de ordinul întâi de aritate n , considerate adevărate.

Reguli

O regulă Prolog exprimă un fapt care depinde de alte fapte și este de forma: $S : -S1, S2, \dots, Sn.$, - and ; - or

- O constantă se referă la un obiect individual. Astfel, în limbajul Prolog ele servesc la definirea obiectelor sau a relațiilor particulare.
- Numerele întregi, numerele reale și șirurile de caractere sunt permise.

Exemple

'ana'

1.2

5

are

mERE

- În Prolog variabilele pot fi înlocuite cu anumite valori. Aceasta înseamnă că unei variabile *i* se poate atribui o anumită valoare sau poate fi legată de ea.

% GRESIT

```
?- X=2, X=X+1, write(X).
```

```
false.
```

% CORECT

```
?- X=2, Y is X+1, write(Y).
```

```
3
```

```
X = 2.
```

```
Y = 3.
```

Atribuirea valorilor pentru variabile

- 1 **=** – este simbolul unificării; în practică $X = a$ înseamnă că X este legat la valoarea a (a este constantă), în timp ce $X = Y$ înseamnă că X și Y sunt legate între ele (Y este o variabilă).
- 2 **is** – denotă atribuire în sensul clasic (se calculează partea dreaptă a expresiei și se atribuie apoi rezultatul variabilei din stânga).

Y is $2 + 1$

IMPORTANT Variabilele care apar în partea dreaptă trebuie să fie instanțiate!

- 1 **Faptele** stabilesc relații între obiectele universului problemei.

Exemple:

```
frumoasa(ana). % Ana este frumoasa.
```

```
bun(vlad). % Vlad este bun.
```

```
cunoaste(vlad, maria). % Vlad o cunoaste pe Maria
```

```
cunoaste(vlad, ana). % Vlad o cunoaste pe Ana.
```

```
iubeste(mihai, maria). % Mihai o iubeste pe Maria.
```

- 2 **Regulile**

```
iubeste(X, Y):- bun(X), cunoaste(X, Y), frumoasa(Y).
```

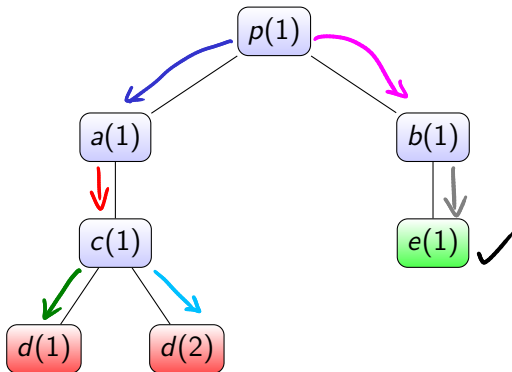
Întrebări asupra unei baze de cunoștințe

$p(1) :- a(1).$
 $p(1) :- b(1).$
 $a(1) :- c(1).$

$c(1) :- d(1).$
 $c(1) :- d(2).$
 $b(1) :- e(1).$

$e(1).$
 $d(3).$

Interogarea: $p(1)$



Întrebări asupra unei baze de cunoștințe

$p(1) :- a(1).$ ✓

$p(1) :- b(1).$ ✓

$a(1) :- c(1).$ ✓

$c(1) :- d(1).$ ✓

$c(1) :- d(2).$ ✓

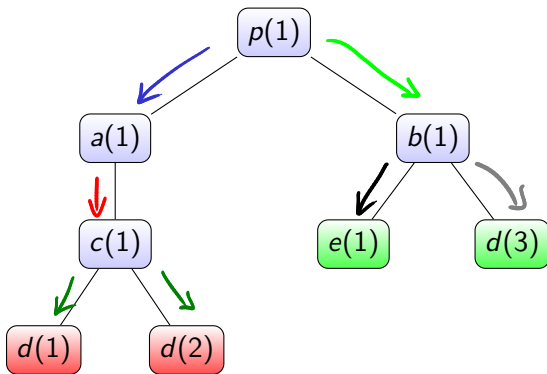
$b(1) :- e(1).$ ✓

$b(1) :- d(3).$ ✓

$e(1).$

$d(3).$

Interogarea: $p(1)$



Întrebări asupra unei baze de cunoștințe

`child_of(john, ann).`

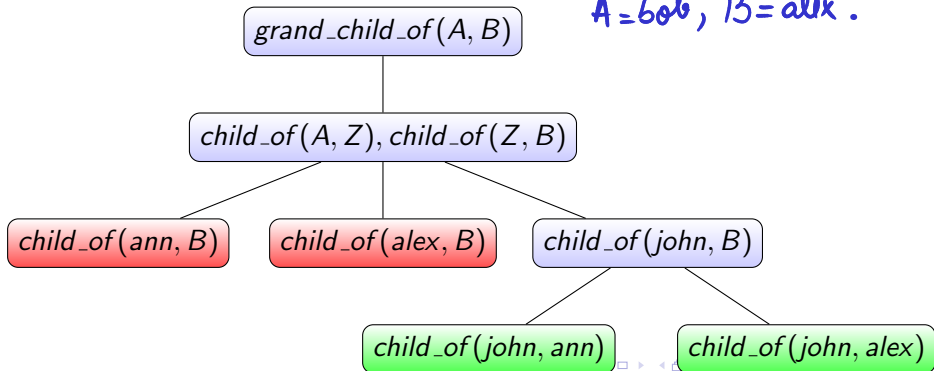
`child_of(john, alex).`

`child_of(bob, john).`

`grand_child_of(X, Y) :- child_of(X, Z), child_of(Z, Y).`

Interogarea: `grand_child_of(A, B).`

*A = bob, B = ann ;
A = bob, B = alex .*



Exemple

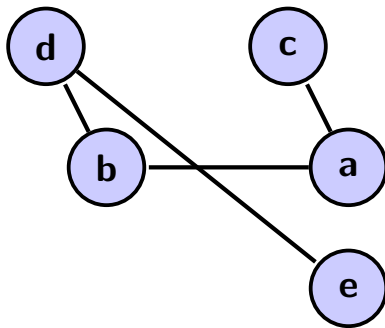
```
persoana(ion).
persoana(ana).
persoana(gheorghe).
persoana(maria).
persoana(marcel).
persoana(andrei).
tata(andrei, ion).
tata(maria, ion).
mama(andrei, ana).
mama(maria, ana).

frate(X, Y) :-
    persoana(X), persoana(Y), X \= Y,
    tata(X, Z), tata(Y, Z), mama(X, W), mama(Y, W).

orfan(X) :- persoana(X), \+tata(X, _), \+mama(X, _).
```

Exemple

```
edge(a,b).  
edge(a,c).  
edge(b,d).  
edge(d,e).
```

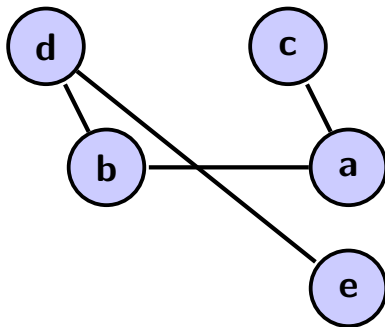


Este corectă varianta?

```
path(X,Y) :- path(X,Z), edge(Z,Y).  
path(X,Y) :- edge(X,Y).
```

Exemple

```
edge(a,b).  
edge(a,c).  
edge(b,d).  
edge(d,e).
```



Este corectă varianta?

```
path(X,Y) :- path(X,Z), edge(Z,Y).  
path(X,Y) :- edge(X,Y).
```

Cazul de bază trebuie să fie adăugat primul!

```
path(X,Y) :- edge(X,Y).  
path(X,Y) :- edge(X,Z), path(Z,Y).
```

Exemple

$\text{Cmmdc}(A, B, C)$

`cmmdc(X, 0, Y) :- X == Y, !.`

`cmmdc(X, Y, C) :- Y > 0, Z is X mod Y, cmmdc(Y, Z, C).`

`fib(0, 0).`

`fib(1, 1).`

`fib(X, Y) :- X > 1, X2 is X - 2, fib(X2, Y2),
X1 is X - 1, fib(X1, Y1),
Y is Y1 + Y2.`

`fib1(0, 0).`

`fib1(X, Y) :- X > 0, fib1(X, Y, _).`

`fib1(1, 1, 0).`

`fib1(X, Y1, Y2) :- X > 1, X1 is X - 1,
fib1(X1, Y2, Y3),
Y1 is Y2 + Y3.`

- **Lista vidă:** $[]$
- Lista cu elementele a, b, c : $[a, b, c]$
- Lista nevidă: $[\text{Prim} | \text{Rest}]$ – unde variabila Prim unifică cu primul element al listei, iar variabila Rest cu lista fără acest prim element
- Lista care începe cu n elemente X_1, X_2, \dots, X_N și continuă cu o altă lista Rest : $[X_1, X_2, \dots, X_N | \text{Rest}]$

$p([H | T], H, T).$

$?- p([a, b, c], X, Y).$

- **Lista vidă:** []
- Lista cu elementele a, b, c : [a,b,c]
- Lista nevidă: [Prim|Rest] – unde variabila Prim unifică cu primul element al listei, iar variabila Rest cu lista fără acest prim element
- Lista care începe cu n elemente X_1, X_2, \dots, X_N și continuă cu o altă lista $Rest$: [X1,X2,...,XN|Rest]

```
p([H|T], H, T).
```

```
?- p([a], X, Y).
```

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

```
X=a
```

```
Y=[b,c]
```

```
true
```

- **Lista vidă:** []
- Lista cu elementele a, b, c : [a,b,c]
- Lista nevidă: [Prim|Rest] – unde variabila Prim unifică cu primul element al listei, iar variabila Rest cu lista fără acest prim element
- Lista care începe cu n elemente X_1, X_2, \dots, X_N și continuă cu o altă lista $Rest$: [X1,X2,...,XN|Rest]

```
p([H|T], H, T).  
?- p([a,b,c], X, Y).
```

```
X=a  
Y=[b,c]  
true
```

```
?- p([a], X, Y).
```

```
X=a  
Y=[]  
true
```

- **Lista vidă:** []
- Lista cu elementele a, b, c : [a,b,c]
- Lista nevidă: [Prim|Rest] – unde variabila Prim unifică cu primul element al listei, iar variabila Rest cu lista fără acest prim element
- Lista care începe cu n elemente X_1, X_2, \dots, X_N și continuă cu o altă lista $Rest$: [X1,X2,...,XN|Rest]

```
p([H|T], H, T).  
?- p([a,b,c], X, Y).
```

```
X=a  
Y=[b,c]  
true
```

```
?- p([a], X, Y).
```

```
X=a  
Y=[]  
true
```

```
?- p([], X, Y).
```

- **Lista vidă:** []
- Lista cu elementele a, b, c : [a,b,c]
- Lista nevidă: [Prim|Rest] – unde variabila Prim unifică cu primul element al listei, iar variabila Rest cu lista fără acest prim element
- Lista care începe cu n elemente X_1, X_2, \dots, X_N și continuă cu o altă lista $Rest$: [X1,X2,...,XN|Rest]

```
p([H|T], H, T).  
?- p([a,b,c], X, Y).
```

```
X=a  
Y=[b,c]  
true
```

```
?- p([a], X, Y).
```

```
X=a  
Y=[]  
true
```

```
?- p([], X, Y).
```

```
false
```

Exemple - Liste

```
quicksort([], []).
quicksort([X|Tail], Sorted) :-
    split(X, Tail, Small, Big),
    quicksort(Small, SSmall),
    quicksort(Big, SBig),
    append(SSmall, [X|SBig], Sorted).
split(_, [], [], []).
split(X, [Y|Tail], [Y|Small], Big) :-
    X >= Y, split(X, Tail, Small, Big).
split(X, [Y|Tail], Small, [Y|Big]) :-
    X < Y, split(X, Tail, Small, Big).
```

Documentarea predicatelor și a argumentelor

- Pentru claritate, antetele predicatelor se scriu sub forma:

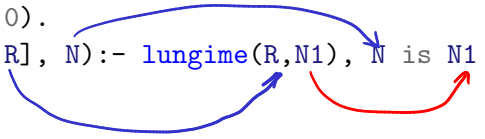
% predicat/nrArgumente

% predicat(+Arg1, -Arg2, ?Arg3, ..., +ArgN)

- Pentru a diferenția intrările (+) de ieșiri(-), se prefixează argumentele cu indicatori. Acele argumente care pot fi fie intrări, fie ieșiri se prefixează cu '?'. Instanțierea parametrilor ține de specificarea acestora:
 - 1 Arg1 va fi instanțiat atunci când se va încerca satisfacerea p/3
 - 2 Arg2 se va instanția odată cu satisfacerea p/3
 - 3 Arg3 va putea fi instanțiat sau nu atunci când se va satisface p/3

Exerciții

```
% lungime(+Lista,-Lungime)
lungime([],0).
lungime([_|R], N):- lungime(R,N1), N is N1 + 1.
```



Exerciții

```
% lungime(+Lista,-Lungime)
lungime([],0).
lungime([_|R], N):- lungime(R,N1), N is N1 + 1.

zip([], [], []).
zip([X|L1], [Y|L2], [(X, Y)|L3]) :- zip(L1, L2, L3).
```

Exerciții

```
% lungime(+Lista,-Lungime)
lungime([],0).
lungime([_|R], N):- lungime(R,N1), N is N1 + 1.

zip([], [], []).
zip([X|L1], [Y|L2], [(X, Y)|L3]) :- zip(L1, L2, L3).

my_last(X,[X]).
my_last(X,[_|L]) :- my_last(X,L).
```

Exerciții

```
% lungime(+Lista,-Lungime)
lungime([],0).
lungime([_|R], N):- lungime(R,N1), N is N1 + 1.

zip([], [], []).
zip([X|L1], [Y|L2], [(X, Y)|L3]) :- zip(L1, L2, L3).

my_last(X,[X]).
my_last(X,[_|L]) :- my_last(X,L).

element_at(X,[X|_],1).
element_at(X,[_|L],K) :- K > 1, K1 is K - 1, element_at(X,L,K1).
```

Exerciții

```
% lungime(+Lista,-Lungime)
lungime([],0).
lungime([_|R], N):- lungime(R,N1), N is N1 + 1.

zip([], [], []).
zip([X|L1], [Y|L2], [(X, Y)|L3]) :- zip(L1, L2, L3).

my_last(X,[X]).
my_last(X,[_|L]) :- my_last(X,L).

element_at(X,[X|_],1).
element_at(X,[_|L],K) :- K > 1, K1 is K - 1, element_at(X,L,K1).

membru(Elem,[Elem|_]).
membru(Elem,[_|Rest]) :- membru(Elem,Rest).
```

Exerciții

```
% lungime(+Lista,-Lungime)
lungime([],0).
lungime([_|R], N):- lungime(R,N1), N is N1 + 1.

zip([], [], []).
zip([X|L1], [Y|L2], [(X, Y)|L3]) :- zip(L1, L2, L3).

my_last(X,[X]).
my_last(X,[_|L]) :- my_last(X,L).

element_at(X,[X|_],1).
element_at(X,[_|L],K) :- K > 1, K1 is K - 1, element_at(X,L,K1).

membru(Elem,[Elem|_]).
membru(Elem,[_|Rest]) :- membru(Elem,Rest).

myConcat([], L, L).
myConcat([H1|T1], L2, [H1|T2]) :- myConcat(T1, L2, T2).
```



Andrei Olaru

Suport de curs - Seria CC

Anul universitar 2014-2015



Mihnea Muraru

Suport de curs - Seria CA

Anul universitar 2014-2015



Mircea Marin

Suport de curs - Programare logica

Anul universitar 2014-2015



Echipa de PP

Breviar laborator

Anul universitar 2015-2016