

1 Your Prolog rules and facts go here ...

# Introducere în Prolog

- Definim fapte și reguli și dăm asta programului logic
- Întrebați-l ce vrem să știm
- Va arăta și va raționa, folosind faptele și regulile disponibile, apoi ne va spune un răspuns (sau răspunsuri)

## Sintaxa

### Atomii

- nume de obiecte sau de predicate
- Încep cu literă mică.
- pot conține în denumire: litere, cifre, \_.
- Orice pus între apostrofuri este tot atom Exemplu: 'don't worry be happy'

### Variabilele

- Încep cu literă MARE
- numele poate conține litere, cifre, \_
- **Variabila anonimă** \_

```
1 whoami([]).
2 whoami([_,_|Rest]):-
3   whoami(Rest).
```

≡ ?- Your query goes here ...

## Prolog reasoning

Considerăm faptul și regula de mai jos:

```
1 ploios(londra).
2 ploios(bangkok).
```

```
3 plictisitor(X):- ploios(X).
```

Putem cere (sau interoga) prolog în promptul său de comandă:

```
≡ ?- % (există un C care face ca acest predicat să fie adevărat?)
```

```
plictisitor(C).
```

Va încerca automat să înlocuiască atomii din fapte în regula sa, astfel încât întrebarea noastră să dea răspunsul adevărat. În acest exemplu:

- Începem cu `plictisitor(X)`, astfel încât programul alege mai întâi un atom pentru X, adică Londra (primul nostru atom din acest exemplu)
- Programul caută să vadă dacă este ploioasă (londra). Există!
- Deci înlocuirea dă rezultatul „adevărat”
- Prologul va răspunde
- C= Londra

Pentru a găsi un **răspuns alternativ**, tastați „;” și „Enter” Va da C= bangkok Dacă nu mai găsește niciun răspuns, va răspunde „nu”

## Example 1

```
1 %facts
2
3 /*clause 1*/ located_in(atlanta,georgia).
4 /*clause 2*/located_in(houston,texas).
5 /*clause 3*/located_in(austin,texas).
6 /*clause 4*/located_in(toronto,ontario).
7
8 /* Rules */
9
10 /*clause 5*/ located_in(X,usa) :-
11     located_in(X,georgia).
12 /*clause 6*/ located_in(X,usa) :-
13     located_in(X,texas).
14
15 /*clause 7*/ located_in(X,canada) :-
16     located_in(X,ontario).
17
18 /*clause 8*/ located_in(X,north_america) :-
19     located_in(X,usa).
20 /*clause 9*/ located_in(X,north_america) :-
21     located_in(X,canada).
```

Pentru a afla (întreba/interoga) dacă **atlanta** se află în **georgia**, vom rula:

```
located_in(atlanta,georgia)
```

Rezultatul va fi **TRUE**

```
≡ ?- located_in(atlanta,georgia).
```



Următoarea interogare primește **FALSE** ca răspuns, deoarece aceasta fapt nu poate fi dedus din cunoștințele din knowledge base. Interogarea reușește dacă primește un **TRUE** și eșuează dacă primește un **FALSE**.

```
≡ ?- located_in(atlanta,texas).
```



## Predicate nedeterminate

Predicatul `located_in` este nedeterminist deoarece poate găsi mai mult de un răspuns.

Prolog poate completa valorile variabilelor.

Există situații în care Prolog nu cere soluții alternative:

1. Interogarea printează un output, iar Prolog deduce că nu se doresc alte soluții.
2. Interogarea nu conține variabile, deci răspunsul returnat va fi **TRUE** sau **\*\*\*FALSE\*\***.

```
≡ ?- located_in(X,texas).
```



```
≡ ?- % exemplu caz 1:  
      located_in(X, texas), write(X).
```



Se pot face interogări pentru oricare argument.

```
≡ ?- % numele regiilor ce contin *austin*  
      located_in(austin,X).
```



```
≡ ?- % orasele din texas  
      located_in(X, texas).
```



```
≡ ?- % toate perechile care pot fi deduse.  
      located_in(X,Y).
```



```
≡ ?- located_in(X,X).
```



Interogarea de mai sus returnează **FALSE** deoarece nu poate deduce o pereche cu aceeași valoare a argumentelor.

# Unificarea și instanțierea variabilelor

- pentru rezolva o interogare:
  - Trebuie să se potrivească (match) cu un fapt sau cu partea stângă a unei reguli (antet).
- **Unificarea variabilelor** = procesul de asignare a unei valori variabilei respective.
- In Prolog, nu există instrucțiune de atribuire
  - Variabilele primesc valori prin potrivire cu constante din fapte sau reguli+ Variabilă **liberă (free)** = variabila care nu are asociată nicio valoare
  - + Variabilă **\*\*legată (bound)\*\*** = variabilă care a primit o valoare
  - O variabilă este legată atât timp cât este necesar pentru a rezolva problema. Apoi, este eliberată (dezlegată) și se caută o altă valoare (=soluție alternativă)
  - Nu se pot stoca informații prin operații de atribuire
  - Variabilele – parte din proces de potrivire, NU tip de stocare de info
    - **Unificare** = proces prin care în Prolog se potrivește o constantă unei variabile

```
1 located_in(X, north_america).
```

\* unifică cu headul propoziției 8 `located_in(X, north_america)` --> `X = austin`. \* body-ul (corpul) propoziției 8 devine noua interogare.

```
≡ ?- trace, located_in(X, north_america).
```

- fiecare instanțiere se aplică DOAR pentru o singură propoziție și doar pentru o invocare a prop. respective.
- X, o dată instanțiat, toți X din aceea clauză vor avea aceeași valoare. \*instanțiere NU Stocare
- instanțiere -> transmitere prin parametru.

```
≡ ?- located_in(austin, S).
```

```
≡ ?- write(S).
```

## Regulile unificării

1. Structuri identice se potrivesc una cu alta
  - `părinte(ana, mihai)` se potrivește cu `părinte(ana, mihai)`.
2. X - variabilă liberă
  - `părinte(ana, X)` se potrivește cu `părinte(ana, mihai)`.
  - Variabila X va fi legată de mihai.
3. X - variabilă legată - se comportă ca o constantă

- Dacă X este legat de mihai:
  - `părinte(ana, X)` se potrivește cu `părinte(ana,mihai)`.
  - `părinte(ana, X)` NU se potrivește cu `părinte(ana,ion)`.
4. Două variabile libere se potrivesc una cu alta:
- `părinte(ana, X)` se potrivește cu `părinte(ana,Y)`.

≡ ?- Your query goes here ...



## Definirea relațiilor

folosind termeni (fapte și reguli) deja definite, putem defini diferite relații între argumente.

```

1 mama(ana, irina).
2 mama(ana, ionut).
3 mama(ana, maria).
4 mama(ana, mihai).
5
6 mama(ioana, ana).
7 mama(ioana, teodora).
8
9 tata(ion, ionut).
10 tata(ion, irina ).
11 tata(ion, maria).
12 tata(ion, mihai).
13 tata(stefan,ion).
14
15 parinte(X,Y):-
16     mama(X,Y).
17 parinte(X,Y):-
18     tata(X,Y).
19
20 frate(X,Y):-
21     mama(Z,X),
22     mama(Z,Y).
23 frate(X,Y):-
24     tata(Z,X),
25     tata(Z,Y).
```



≡ ?- parinte(X,Y).



- programul va căuta prima regulă.
- Dacă aceasta nu este ok fie:
  - caută o implementare alternativă a regulii
  - se întoarece(prin backtracking) și încearcă a doua regulă.

# Conjunția predicatelor

Presupunem că dorim să găsim tatăl lui **ion** și pe tatăl tatălui lui **ion**.

```
≡ ?- tata(X,ion), tata(X,F).
```



Dacă apelăm predicatele în ordine inversă vom obține:

```
≡ ?- tata(X,F), tata(X,ion).
```



## disjuncția predicatelor

- In Prolog, caracterul ; există și este semnul de disjuncție (SAU).
- deseori, caracterul ; poate fi confundat cu caracterul ,
- este recomandat să se folosească implementări diferite pentru același predicat, mai degrabă decât disjuncția.

## goaluri negative ("NOT")

- \+ se citește: "not " sau "nu se poate demonstra".
- poate fi apelat asupra oriărui goal.
- dacă g este u goal care reușește, atunci \+g va returna fail.
- dacă g este u goal care dă fail, atunci \+g va reuși.

```
≡ ?- tata(ion, mihai).
```



```
≡ ?- \+ tata(ion, mihai).
```



```
1 non_parent(X,Y):-
2   \+ father(X,Y), \+ mama(X,Y).
```



X este considerat că nu este părintele lui Y dacă nu putem demonstra că X este tatăl lui Y și nu putem demonstra că X este mama lui Y.

## Modele de flux

Legările de variabile: la intrare sau la ieșire din clauză

**Model de flux** = direcția în care se leagă o variabilă

**Parametru de input** = variabilă data la intrare

**Parametru de output** = variabila este dată la ieșire

Exemplu: predicatul `factorial(n,f)` .

1. (l,i) - se verifică dacă  $n! = f$
1. (l,o) - atribuie  $f:=n!$
2. (o,i) - găsește acel  $n$  pentru care  $f= n!$

Un predicat poate avea mai multe modele de flux

## Operații aritmetice în Prolog

1. Expresii matematice
2. Operatorul **is** - evaluare expresii aritmetice
  - Argumentul din dreapta trebuie să fie o expresie aritmetică cu variabile instanțiate
  - Argumentul din stânga - număr sau variabilă reprezentată ca număr
  - `True =>` rezultatul expresiei din dreapta = valoarea argumentului din stânga
  - `Swi-Prolog =>` `float + integer = float`.

≡ ?- 1.0 is 0.5+0.5



≡ ?- 1 is 0.5+0.5.



## Relații matematice predefinite

- Folosite pentru a compara 2 numere
- Diferența dintre `=`, `is` și `==`
  - operatorii `==` și `is` forțează evaluarea unei expresii
  - operatorul `=` verifica doar egalitatea structurală.

≡ ?- 3+5=8.



≡ ?- 3+5 ==8.



## Operatori în Prolog

- Scrieri echivalente:
  - `100 + 1 <=> +(100,1)`
  - `A is B*3 <=> is(A (B+1))`
  - Precedența operatorilor = Ordinea efectuării operațiilor
    - Nr între 0-1200
    - The lower the precedence – the stronger binding

- '\*' are precedență 400
- '+' are precedență 500
  - $1+2*3 = 1+ (2*3)$

Asociativitatea operatorilor – parte din definiție

Precedența operatorilor uzuali o găsiți aici (<https://ocw.upj.ac.id/files/Textbook-TIF212-Prolog-Tutorial-3.pdf>).

≡ ?- current\_op(Precency, Associativity, -).



Empty markdown cell. Double click to edit

- Predicate – definite ca funtori : hrănește(câine, os) .
  - Predicate – definite ca operatori:
    - Infixat (functor scris între argumente, fără paranteze) - xfy
      - câinele (se) hrănește (cu) os
    - Prefixat (functor scris înaintea argumentelor, fără paranteze) -fy
      - hrănește câinele (cu) os
    - Postfixat (functor scris după argumente, fără paranteze) - xf
      - câinele (cu) os (se) hrănește
- Cod mai ușor de citit

```
1 is_bigger(elephant,mouse).
2 is_bigger(dog,cat).
3
4 ?-op(20,xfy,is_bigger).
```



≡ ?- dog is\_bigger cat.



```
1 este_foame(ion).
2 este_dulce(bomboane).
3
4
5 mananca(ion,Y):-
6     este_dulce(Y),
7     este_foame(ion).
8
9 ?-op(40, xfy, mananca).
10 ?-op(30, xf, este_dulce).
11 ?-op(30, xf, este_foame).
12
13 ion mananca Y :- Y este_dulce, ion este_foame.
```



≡ ?- ion mananca Y.





# Manipularea listelor în Prolog

Descompunere [Cap|Coadă]

- Accesarea primului element
- Accesarea celui de-al doilea element
  - Ex.: Concatenarea a două liste: (hands on)

```
1 %model matematic:
2 %concatLists(l1,..ln, k1,..km)= {k1,...km,    L=[],
3 %                               {l1 U concatLists(l2,..ln, k1,...km),
4 %
```

≡ ?- Your query goes here ...

## Liste eterogene și liste omogene

**Liste omogene** = liste formate doar din atomi și numere

**Liste eterogene** = Liste formate din atomi, numere și subliste.

MOD DE LUCRU

- Se descompune lista în Cap (H) și Coadă (T): [H|T]
- Se verifică tipul elementului Coadă - poate fi atom, număr sau listă
  - `is_list(H)` - returnează true dacă H este o listă
  - `number(H)` - - returnează true dacă H este un număr
  - `atom(H)` -- returnează true dacă H este atom (simbol)