

||LAB 10: INTRO PROLOG||

SINTAXA & SEMANTICA

- clauze / predicate [Japte
reguli]
- constante = atomi (cu literă mică)
- variabile (- cu literă mare)
- structuri
- operatori
- tipuri de date - liste, siruri

① AXIOME / FACTS

- sunt necondiționat / mireu adverat
 - stabilește "relații" între obiecte

Sintaxă: relation(object₁, object₂, ...).

$E_x :$

fruct (mar).
albare (rosu).
om (yach). } predicate
mare

Q RULES

→ relații între obiecte adică variabile
condiționate

Întrată : $P ::= P_1, P_2, \dots, P_n$

Head neck body

(\Leftarrow) (\Leftarrow)

";" = sau ";" = și

\Leftarrow Body $\in \Theta \Rightarrow$ Head $\in \Theta$

$(P_1, P_2, \dots, P_n \text{ sunt } \Theta \Rightarrow P \Theta)$

Ex:

Mary e fericit dacă bea cafea
și mananca luisante.

fericit (...); bea (...), ...)
mananca (...), ...)

fericit (mary) :- bea(mary, cafea),
mananca (mary, luisante).

③ QUERIES

→ Mărebiri despre lăptă / relații

Ex:

- Este marel un fruct?
! - fruct(mare).
- Cineva sănătate și John practicează?
! - practiceaza(sănătate, John).

IPOTEZA LUMII RICHISE

KNOWLEDGE BASE = colectiv

facts & rules

! Când se face un query, Prolog cauta în KB

nu găsește ceea ce => false

④ VARIABLES

- cu literă mare
- pt. a generaliza reguli

Ex:

"Un om este fericit dacă
bea cafea și manancă băcălie."

fericit(x) :- om(x), bea(x , cafea),
mananca(x , băcălie).

→ În întrebare: X dă valori
variabilelor pe baza KB pt. a
satisfac predicatul.

④ PROCESUL DE EXECUȚIE

→ se parcurează clauzele din dreapta de la stanga la dreapta și se mărcă satisfacerea lor.

Ex:

```
om(mary).  
om(helen).  
om(john).  
  
bea(mary, cafea).  
bea(mary, ceai).  
bea(helen, ceai).  
bea(john, cafea).  
  
mananca(mary, ciocolata).  
mananca(helen, ciocolata).  
mananca(helen, biscuiti).  
mananca(john, alune).  
  
fericit(X) :- om(X), bea(X, cafea), mananca(X, ciocolata).
```

Achivăm trace ca să vedem
ce pasi executa Prolog la query-ul
"fericit(X)."

```

[trace] 2 ?- fericit(X).
Call: (10) fericit(_17214) ? creep
Call: (11) om(_17214) ? creep
Exit: (11) om(mary) ? creep
Call: (11) bea(mary, cafea) ? creep
Exit: (11) bea(mary, cafea) ? creep
Call: (11) mananca(mary, ciocolata) ? creep
Exit: (11) mananca(mary, ciocolata) ? creep
Exit: (10) fericit(mary) ? creep
X = mary ;
Redo: (11) bea(mary, cafea) ? creep
Fail: (11) bea(mary, cafea) ? creep
Redo: (11) om(_17214) ? creep
Exit: (11) om(helen) ? creep
Call: (11) bea(helen, cafea) ? creep
Fail: (11) bea(helen, cafea) ? creep
Redo: (11) om(_17214) ? creep
Exit: (11) om(john) ? creep
Call: (11) bea(john, cafea) ? creep
Exit: (11) bea(john, cafea) ? creep
Call: (11) mananca(john, ciocolata) ? creep
Fail: (11) mananca(john, ciocolata) ? creep
Fail: (10) fericit(_17214) ? creep
false.

```

} găsește legarea
 $X = \text{mary}$ și
 predicatul
 $\text{fericit}(X)$ este $\text{A}.$
 Verifică și restele
 posibilităților,
 dar nu se dă
 fail și împarcă
 False

OBS: • Prolog marchează legarea lui
 X la constante care satisfac
 predicatelor din dreptă regulă;
 • Prolog face backtracking în
 KB pentru a verifica dacă
 posibilitatele de satisfacere a predicatelor.

⑤ NEGATIA GA EJEC

→ " \+ " : nu e ac. lucru cu
niciunul din alte limbișe

→ \+ verifica :

I dăcă un scop pe care fi satisfacut

student(mary).

student(helen).

lazy(mary).

$\rho(x) : -\underline{\text{student}(x)}, \underline{\text{\+ lazy}(x)}$.

. - $\rho(x)$.

$X = \underset{\textcircled{1}}{\text{mary}}, \underset{\text{true}}{\text{\+ lazy(mary)}} \Rightarrow \textcircled{F}$

$X = \underset{\textcircled{1}}{\text{helen}}, \underset{\text{false}}{+\text{\+ lazy(helen)}} \Rightarrow \textcircled{T}$

II dacă continu variabilele verifică
dacă și legătu r atât mult
predicatul e adevărat.

student(mary).

student(Helen).

lazy(mary).

$\rho_2(x) :- \underline{\neg + \text{lazy}(x)}, \text{student}(x).$

! - $\rho_2(x)$.

! găsește $x = \text{ana}$ pt. care $\text{lazy}(\text{ana}) \text{ A}$

$\Rightarrow \neg + \text{lazy}(x)$ întreacă False

⑥ OPERATORI

→ control : " ; ' ; ' , ' \ + ' .

→ aritmetică : + , - , * , /

→ relationale aritmética : = \ = ,
, < , > , = < , > = , = : = , is

→ unificare : = , \ = , = \ = , \ == .

UNIFICARE = procesul de identificare
a valorilor variabilelor din cel
putin două expresii, astfel încât
să se substituie variabilelor prin valorile
alese să ducă la coïncidență
expresiilor.

• " = " + opusul " \="

→ operatorul de unificare

Ex :

$$1+2 = 1+2 \rightarrow \text{true}$$

$$1+2 = 2+1 \rightarrow \text{false}$$

$$X = 1+2 \rightarrow X = 1+2 \quad (\text{unificare})$$

! fără calcul

$$1+X = 1+2 \rightarrow X = 2$$

$$2+X = 1+2 \rightarrow \text{false}$$

$$\underline{2+X} \backslash = 1+2 \rightarrow \text{true}$$

$$1+X = y+2 \rightarrow X = 2; y = 1.$$

$$1+X = 1+2+3 \rightarrow \text{false}$$

• " == " + opusul " \== "

→ verifică dacă doi operandi sunt același lucru

Ex:

$$1 + 2 == 1 + 2 \rightarrow \text{true}$$

$$1 + 2 == 2 + 1 \rightarrow \text{false}$$

$$1 + 2 == 1 + X \rightarrow \text{false}$$

(fără unicare)

- "is" \rightarrow se va evalua să opereze
din dreapta

Ex:

$$3 \text{ is } \frac{1+2}{3} \rightarrow \text{true}$$

$$1+2 \text{ is } \frac{1+2}{3} \rightarrow \text{false } (1+2 \neq 3)$$

$$X \text{ is } \frac{1+2}{3} \rightarrow X = 3 \text{ (legare)}$$

$$1+2 \text{ is } X \rightarrow \text{error } (X=?)$$

• " = ; = " + operuel " = ~ = "

→ evaluarea ambeui operant
și le compara rolurile

Ex:

$$1 + 2 = : = 1 + 2 \rightarrow \text{free}$$

$$2 + 1 = : = 1 + 2 \rightarrow \text{free}$$

$$x = : = 1 + 2 \rightarrow \text{error}$$

(x = ?)

7 LISTE

→ Lista nuda: []

→ Lista cu elemente: [1, 2, 3]

→ Lista nevăzătoare: [H | T]

→ Lista cu cel puțin 2 elemente
[H1, H2 | T].

Ex:

0 Lungimea unei liste

```
/*
functie pentru a calcula lungimea unei liste
my_length(+list, -length)
+Lista input
-Lungime output
```

```
{ Haskell:
length [] = 0
length (_ : rest) = length(rest) + 1
*/
```

```
my_length([], 0).
my_length([_ | Rest], Len) :- my_length(Rest, Len1), Len is Len1 + 1.
```

la z de baza
aici se depune lungimea listei

se merge în recursivitate
pentru a calcula lungimea restului listei "Rest"
odata cu len! este calculat, și calculata și len

Convenție în documentație:

• + : intrare

• - : ieșire

• ? : intrare / ieșire

Q verificare dacă un element
se află într-o listă

```
/*
    my_member(+X, +L)

    { member(x, []) = false
      member(x, (y : rest)) = x == y | member(x, rest)
    }

% my_member(_, []):- false. ①
my_member(X, [X | _]). ②
my_member(X, [_ | T]) :- my_member(X, T). ③
```

Observații:

① = cazul de bază returnând
valoarea *false*, deci poate fi omis.

De ce! - Prolog funcționează pe
"protectă lumii" închise", dacă dice
căz căre nu e definită să
considerat fals

Q = Cazul în care cor. elementul
cautat este egal cu head-ul listei.
a) se întoarcere **free**.

③ = Dacă nu s-a făcut
match pe cazul ②, se continuă
cautarea în restul listei.