PROGRAMARE LOGICA

Programare in PROLOG

Avantajele limbajelor de programare logica

- Reprezentare simbolica
- Stil declarativ: spunem CE stim despre problema, nu CUM se ajunge la solutie
- Separarea datelor de procesul de inferenta (care e incorporat in limbaj)
- Mod uniform de a reprezenta faptele si regulile
- Fiecare fapt/regula se reprezinta printr-o propozitie distincta => buna modularizare
- Forma relationala a regulilor face posibil ca acestea sa fie reversibile (sa functioneze si in sens invers, pornind de la rezultat)

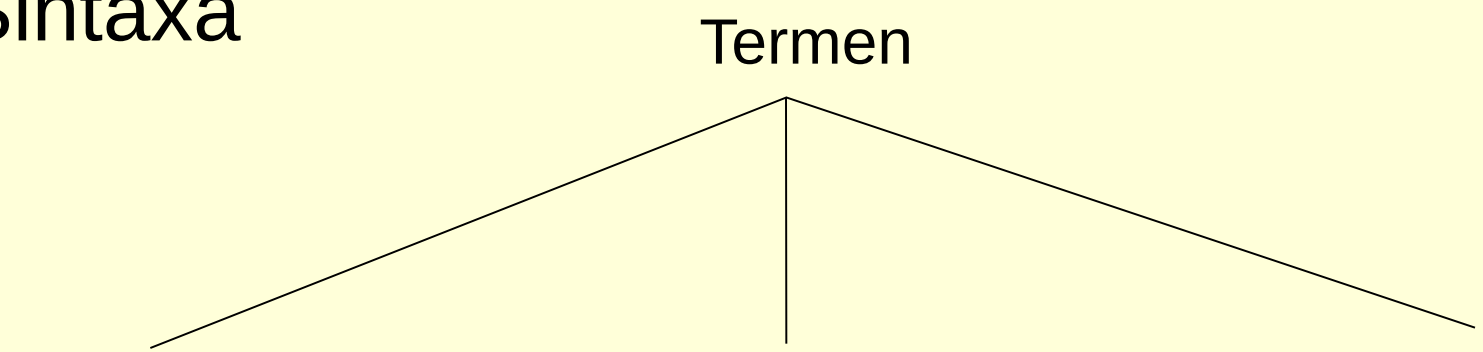
PROLOG

- FOL + restrictii
- Demonstratiile se realizeaza prin reducere la absurd (bazat pe aplicarea rezolutiei)
- Backward chaining+DFS pentru controlul pasilor de demonstratie (solutia se gaseste cautand peste tot in spatiul solutiilor posibile)
- Procedeu de baza in manipularea termenilor: unificarea
- Variabile = “goluri” in structurile de date, care se umplu pe masura ce sunt necesare noi unificari

Restrictii

- Clauze = clauze Horn
  $P1 \wedge P2 \wedge \ldots \wedge Pn \Rightarrow P$
- Nu permite unificare ciclica
- Nu permite negatia logica
  not(P) = “in programul curent, P nu poate fi demonstrat”
- Ipoteza lumii inchise
  adevaruri exista doar in program; orice din afara este fals

Sintaxa



Constanta

obiect

Atom

identificator care incepe cu litera mica

peter_pan
tinkerbell

Numar

intreg/real

0
-1
3.14

Structura

obiect cu componente

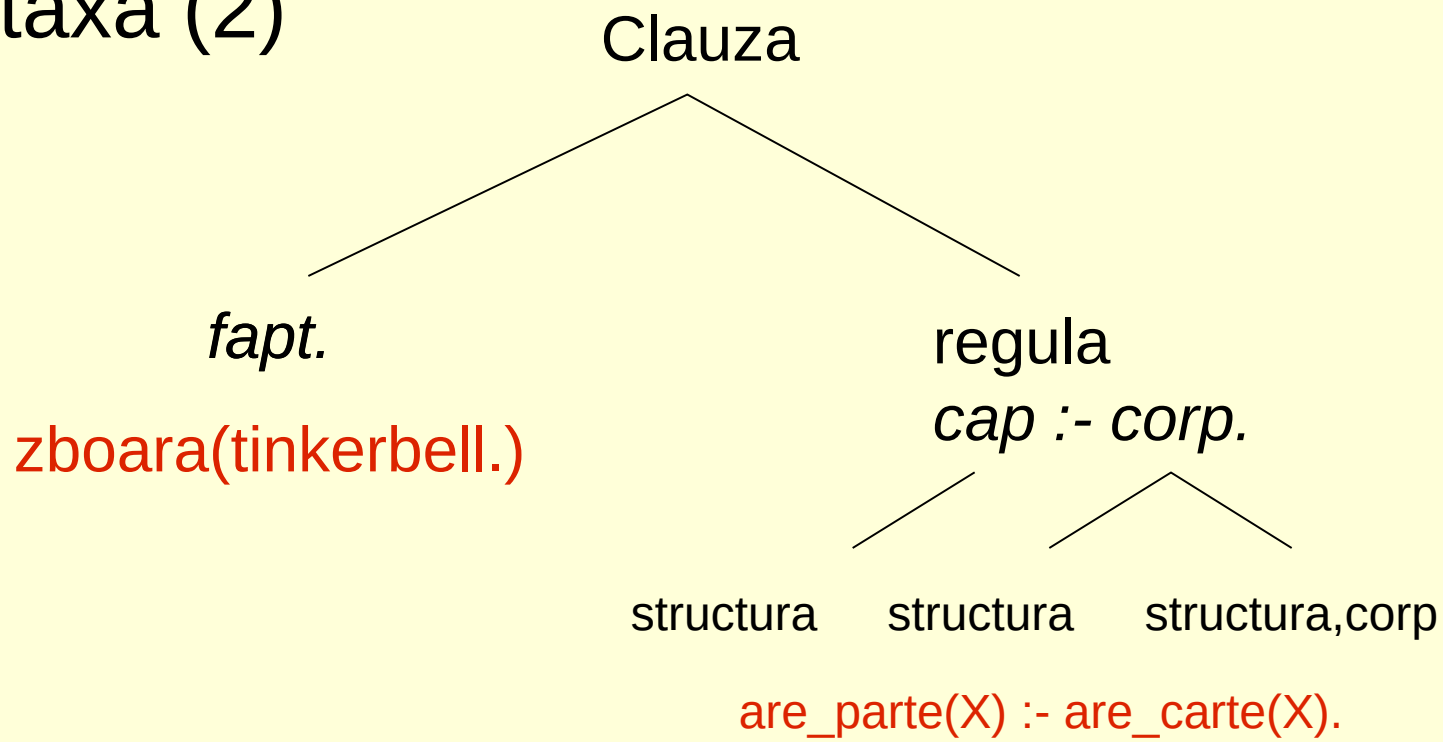
cadou(carte(aventuri), stilou,minge(rugby))

Variabila

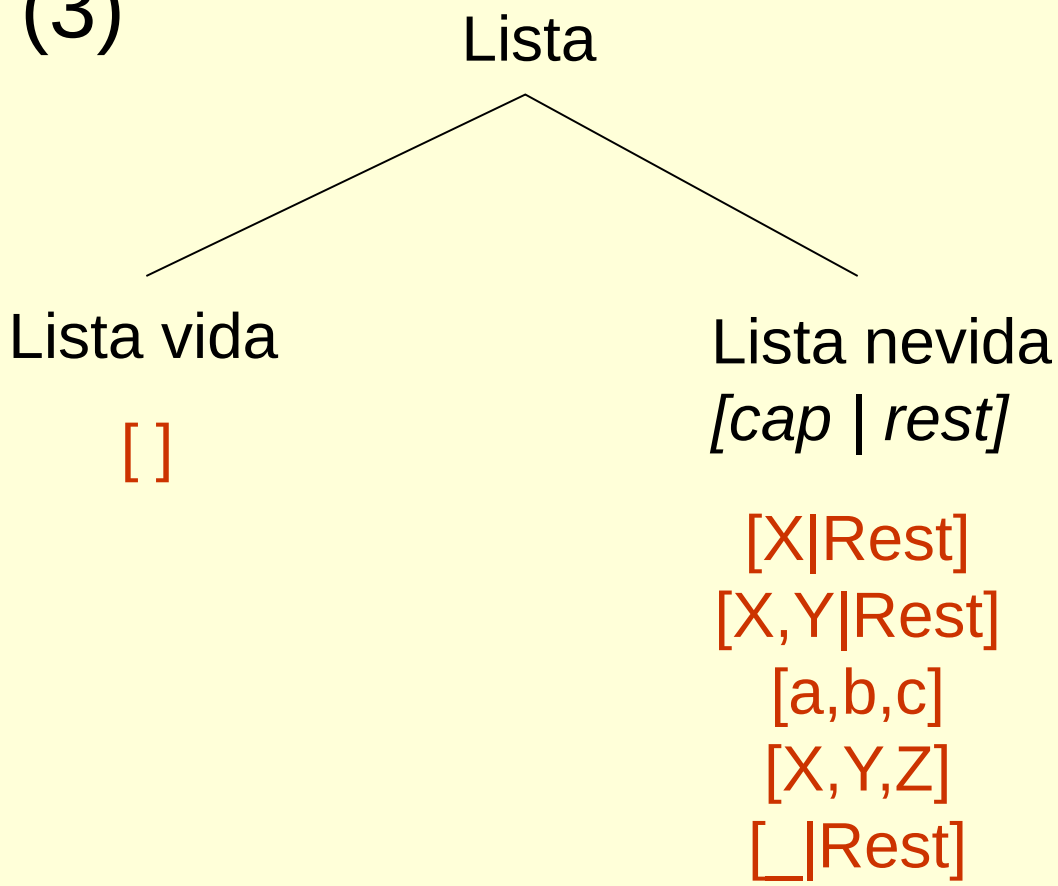
obiect care nu poate fi identificat in momentul scrierii programului

X
O_O
_

Sintaxa (2)


Clauza
fapt.
zboara(tinkerbell.)
regula
cap :- corp.
structura
structura
structura,corp
are_parte(X) :- are_carte(X).

Sintaxa (3)


Lista
Lista vida
[ ]
Lista nevida
[cap | rest]
[X|Rest]
[X,Y|Rest]
[a,b,c]
[X,Y,Z]
[_|Rest]

Operatori

- Aritmetici predefiniti:
  + - * / mod
- Relationali predefiniti
  = \= < > =< >= =:= =\=
- Operatorii sunt functori: 1+2 = 1+2; 1+2 ≠ 3!
- Evaluarea aritmetica se face la cerere, folosind *is*
  modulo(X,Y,X) :- X<Y.
  modulo(X,Y,Z) :- X>=Y, X1 is X-Y, modulo(X1,Y,Z).

Query-uri

- Moduri de a chestiona programul despre ce este adevarat

?- zboara(tinkerbell).

true.

?- zboara(X).

X = tinkerbell;

false.

Comparatie imperativ/functional/declarativ

```
append(A,B) {
 C=copy(A)
 while C.tail!=nil
   C=C.tail
 C.tail=B
 return C
}
```

```
append(A,B) =
  if null(A)
  then B
  else cons(head(A),append(tail(A),B))
```

```
append([ ],B,B).
append([X|Rest],B,[X|R]) :- append(Rest,B,R).
```

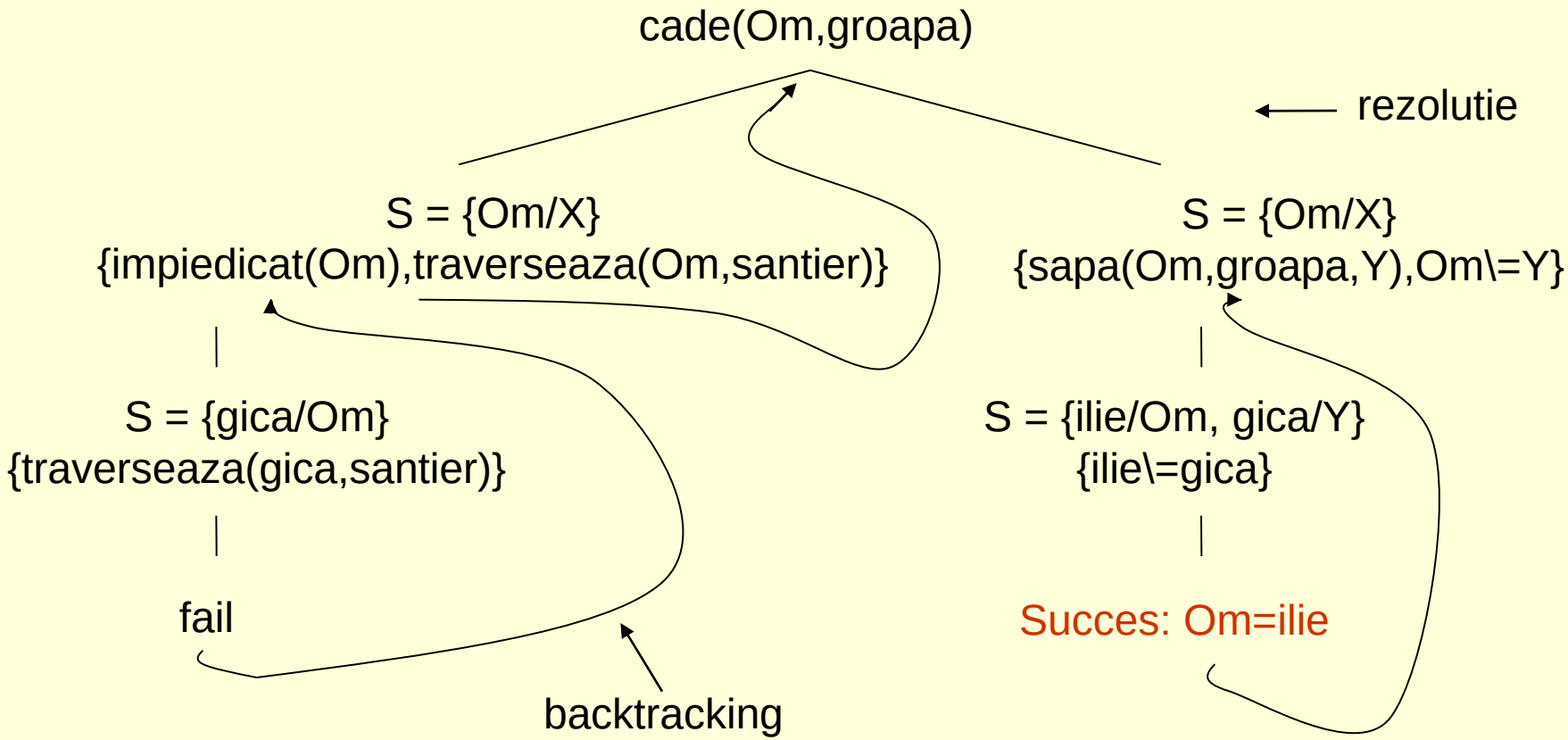
Exemplu (de functionare a unui program Prolog)

```
cade(X,groapa) :- impiedicat(X), traverseaza(X,santier).
cade(X,groapa) :- sapa(X,groapa,Y), X\=Y.

impiedicat(gica).
traverseaza(ilie,santier).
sapa(ilie,groapa,gica).

?- cade(Om,groapa).
```

Exemplu (de functionare a unui program Prolog) (2)



Observatii

- Conteaza ordinea clauzelor.
- Similar, conteaza ordinea premiselor in corpul clauzei.
- Ca regula generala, ce e mai simplu de satisfacut se pune mai intai (faptele inainte de reguli, iesirile din recursivitate inainte de apelurile recursive etc).

Strategii de control

- Forward chaining (data driven)
  - se genereaza toate propozitiile care se pot genera din Descr, oprind procesul in caz ca se ajunge la scop

- Backward chaining (goal driven)
  - inferenta se concentreaza doar pe regulile care pot duce la derivarea scopului
  - se pleaca de la scop si se cauta regulile a caror concluzie unifica cu scopul
  - pentru fiecare din ele, pe rand, premisele ei se adauga la scopuri si se reia procesul de implinire a noilor scopuri

Forward versus backward chaining

- Cele 2 strategii rezolva aceleasi probleme, insa trebuie ales intre ele in functie de particularitatile problemei

- Forward: cand sunt foarte multe scopuri (foarte multe posibile raspunsuri corecte) (exemplu: posibile configuratii pentru o masina)
- Backward: cand sunt putine scopuri (exemplu: caut un diagnostic si am cateva variante despre de ce ar suferi pacientul)

Algoritmul pentru motorul de inferenta din Prolog

```
backward_chaining(Clauze,Scopuri,Legari) {
    if(Scopuri=[ ]) return success
    scop=head(Scopuri); Scopuri=tail(Scopuri)

    for_each(clauza in Clauze) { //in ordine
      if(unifica(scop,antet(clauza),Legari,NoiLegari)) {
            L = Legari U NoiLegari
            S = append(corp(clauza),Scopuri) // DFS
            if(backward_chaining(Clauze,S,L)) return success
      }
    }
    return fail
}
```

Mecanisme de control in limbaj

Predicate

- true – reuseste intotdeauna
- fail – esueaza intotdeauna
- ! (se citeste “cut”):
  - prima data, cut reuseste
  - cand revin din backtracking la cut, acesta esueaza
  - toate regulile urmatoare al caror antet unifica cu antetul regulii cu cut – sunt ignorate!

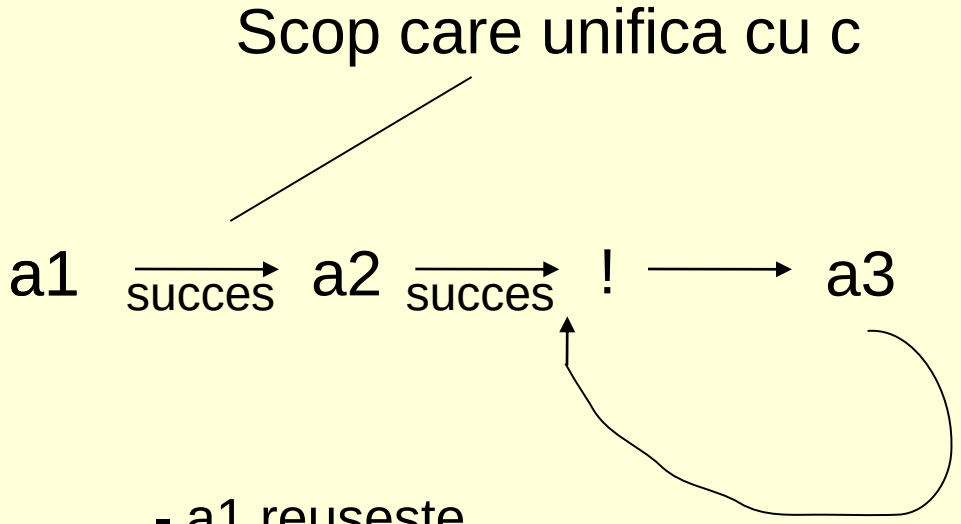
Exemplu cu cut

not(P) :- P,!,fail.

not(P) :- true.

c :- a1,a2,!,a3.

c :- b1,b2.



- a1 reuseste
- a2 reuseste
- cut reuseste
- trece la a3 care reuseste de n ori
- bkt-ul revine la cut care esueaza
- nu se mai pot resatisface a1, a2