

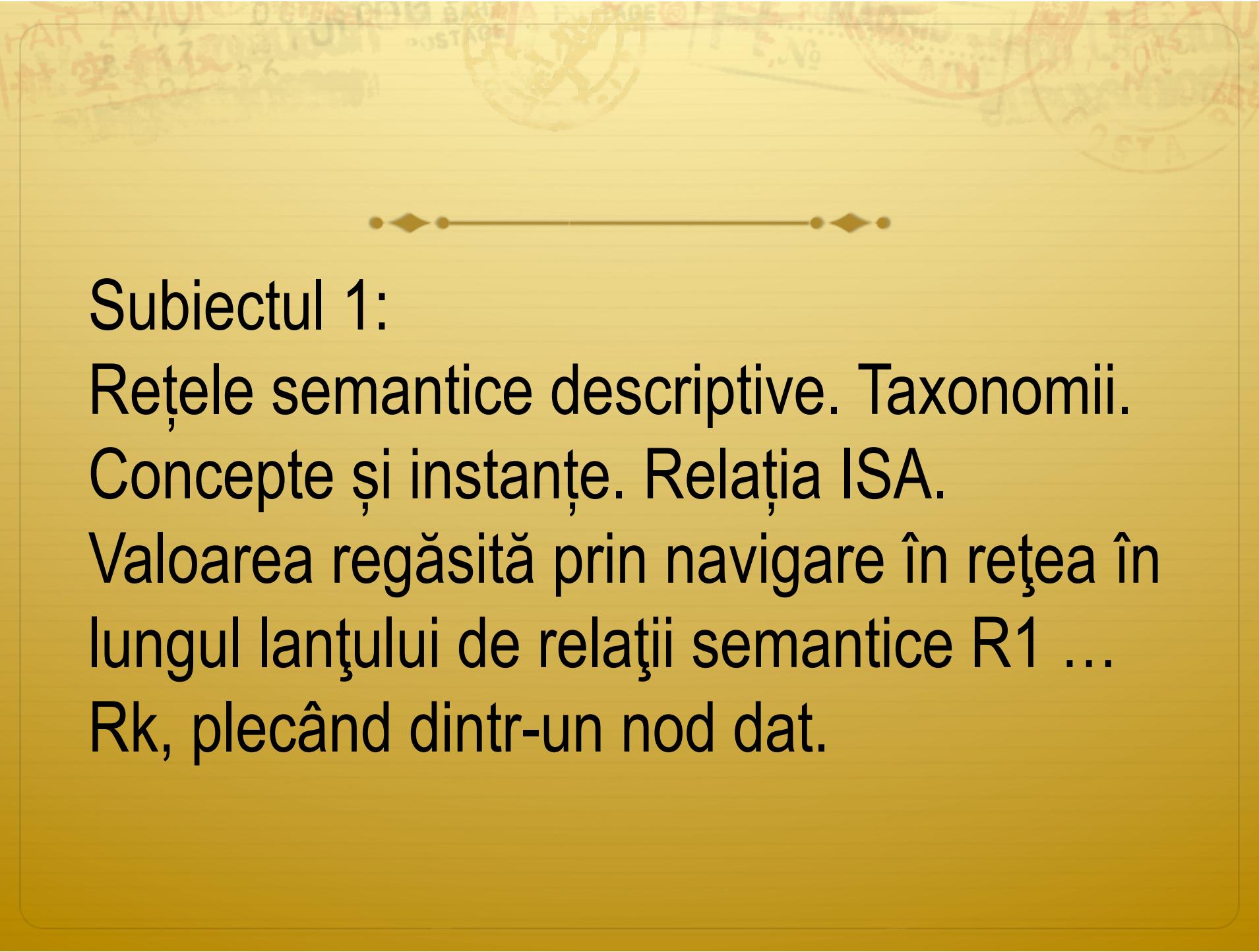


Răspunsuri la subiectele de IA

Sesiunea de licență 2018

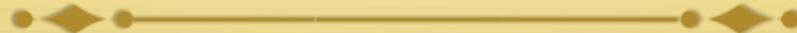


Dan Cristea, Ionuț Pistol, Diana Trandabăț, Mădălina Răschip



Subiectul 1:
Rețele semantice descriptive. Taxonomii.
Concepțe și instanțe. Relația ISA.
Valoarea regăsită prin navigare în rețea îl
lungul lanțului de relații semantice R1 ...
Rk, plecând dintr-un nod dat.

Clase și instanțe



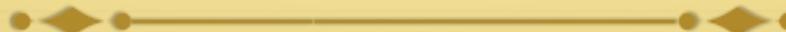
- O clasă (concept): un set care conține indivizi
- Un individ (instanță) poate apartine la mai multe clase
- O ierarhie superclasă-subclasă se numește taxonomie

<Declaration>

```
  <Class IRI="#corp-geometric" />
```

</Declaration>

Rețelele semantice descriptive permit reprezentarea economică



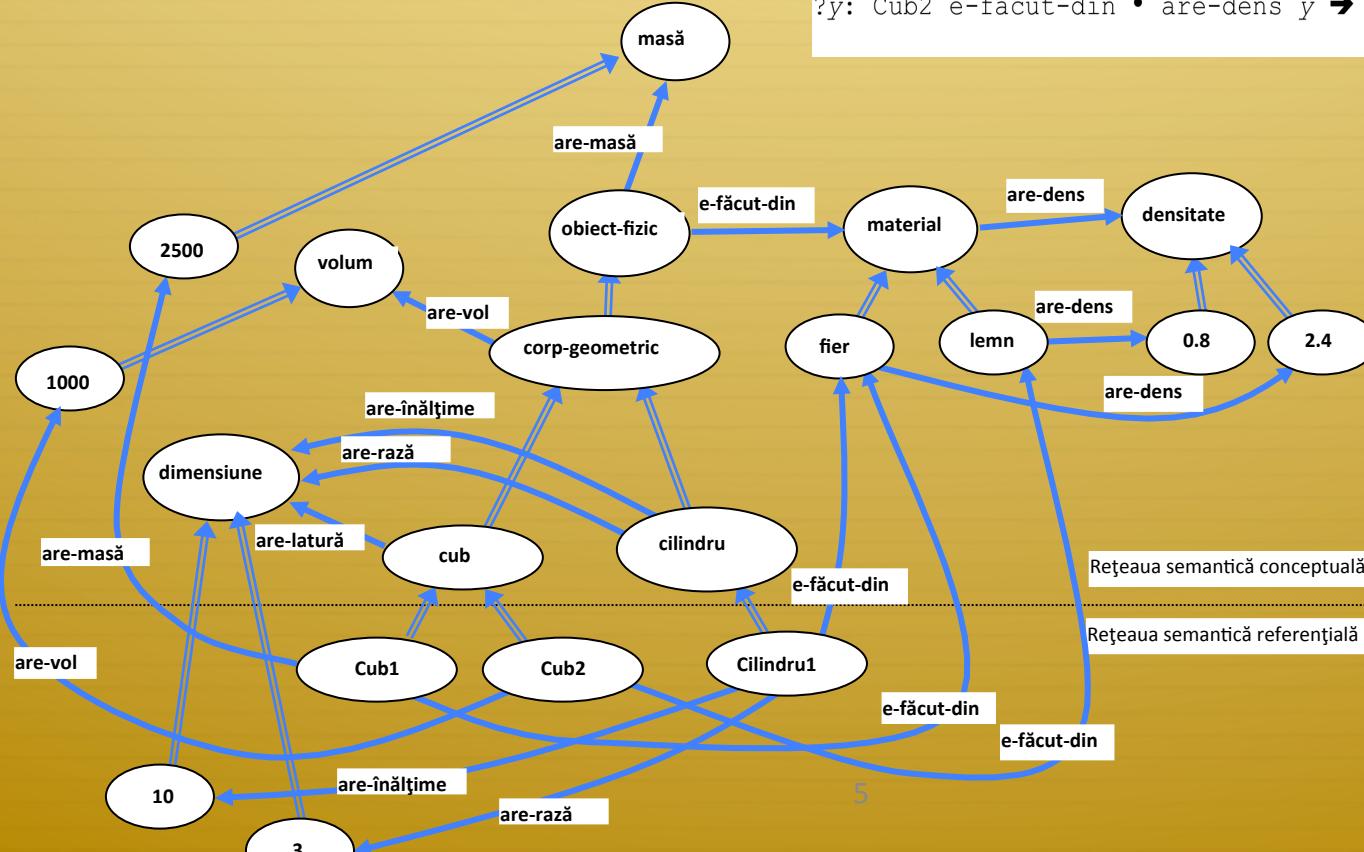
- ❖ Proprietăți:
 - ❖ explicite – la nivelul conceptual
 - ❖ implicite (moștenite) – la nivelul referențial
- ❖ Interogări:
 - ❖ care este închiderea tranzitivă a relațiilor taxonomice ISA ale unui nod din rețea?
 - ❖ ce valoare este atașată prin relația semantică R nodului n ?
 - ❖ care este valoarea regăsită prin navigare în rețea îngustă lungul lanțului de relații $R_1 \dots R_n$, plecând din nodul n ?
 - ❖ care este calea de relații semantice ce se poate stabili între două noduri n_1 și n_2 ?

Interrogări într-o rețea semantică



Care este densitatea corpului Cub2?

? C_{Cub2} : Cub2 ISA $C_{Cub2} \rightarrow C_{Cub2} = \text{cub}$
?R*: cub R* densitate $\rightarrow R^* = e\text{-făcut-din} \bullet \text{are-dens}$
?y: Cub2 e-făcut-din • are-dens y $\rightarrow y = 0.8$

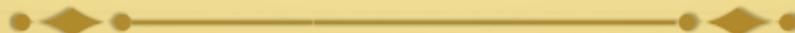




• ◆ • ————— • ◆ •

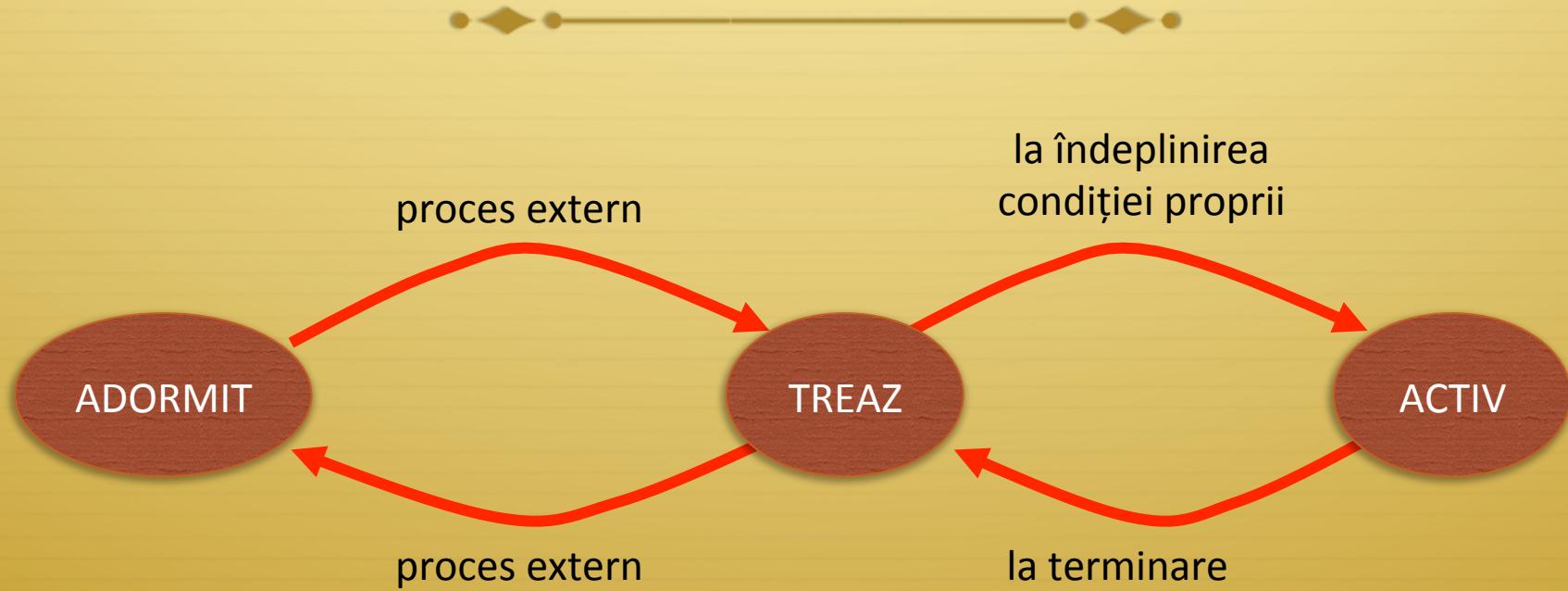
Subiectul 2: Rețele semantice descriptive. Demoni. Exemple.

Demoni

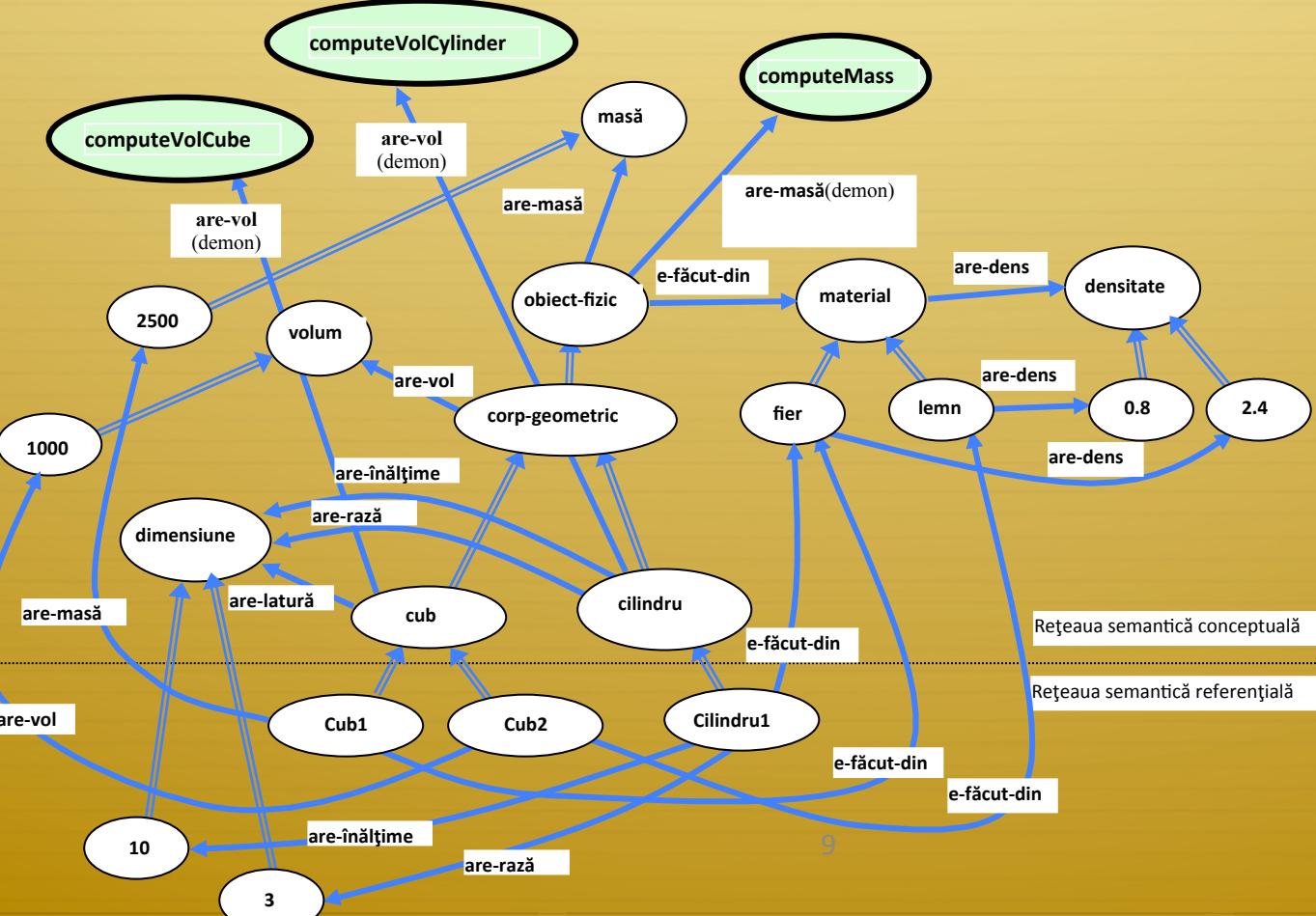


- ❖ Proceduri care...
 - ❖ nu se apelează
 - ❖ se activează singure când anumite condiții pe care ei sunt pregătiți să le sesizeze sunt îndeplinite
- ❖ Stările unui demon:
 - ❖ **adormit**
 - ❖ **disponibil** (*idle*)
 - ❖ **activ**

Tranzițiile demonilor



Demoni într-o rețea semantică

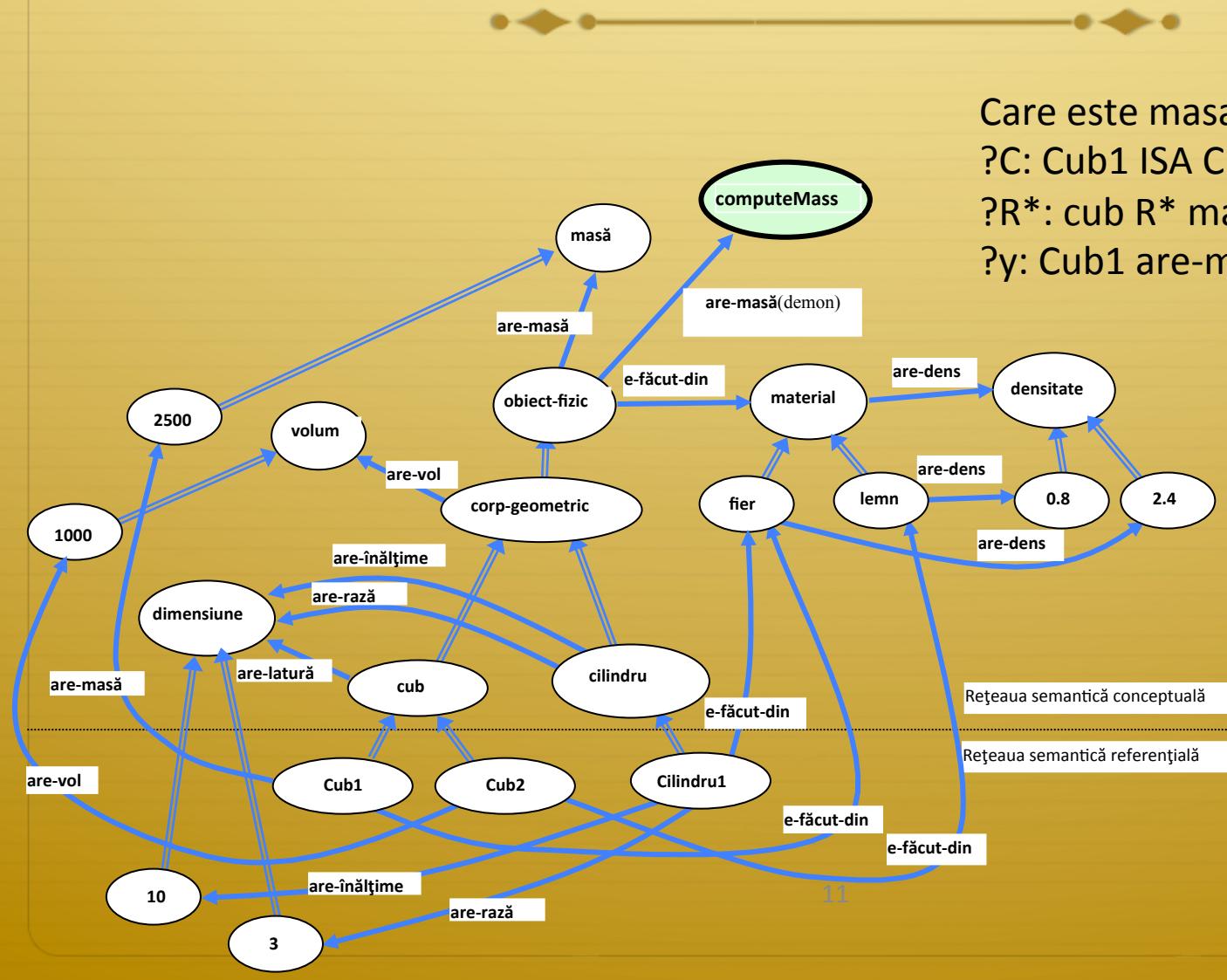


Demonul ComputeMass

$$m = \rho * V$$

```
procedure ComputeMass( x )
begin
; află densitatea lui x:
?Cx: x ISA Cx
?R1*: Cx R1* densitate
?y1: x R1* y1
; află volumul lui x:
?R2*: Cx R2* volum
?y2: x R2* y2
; calculează masa ca densitate * volum:
  return y1 * y2;
end
```

Activarea demonilor (demonul nu se activează)



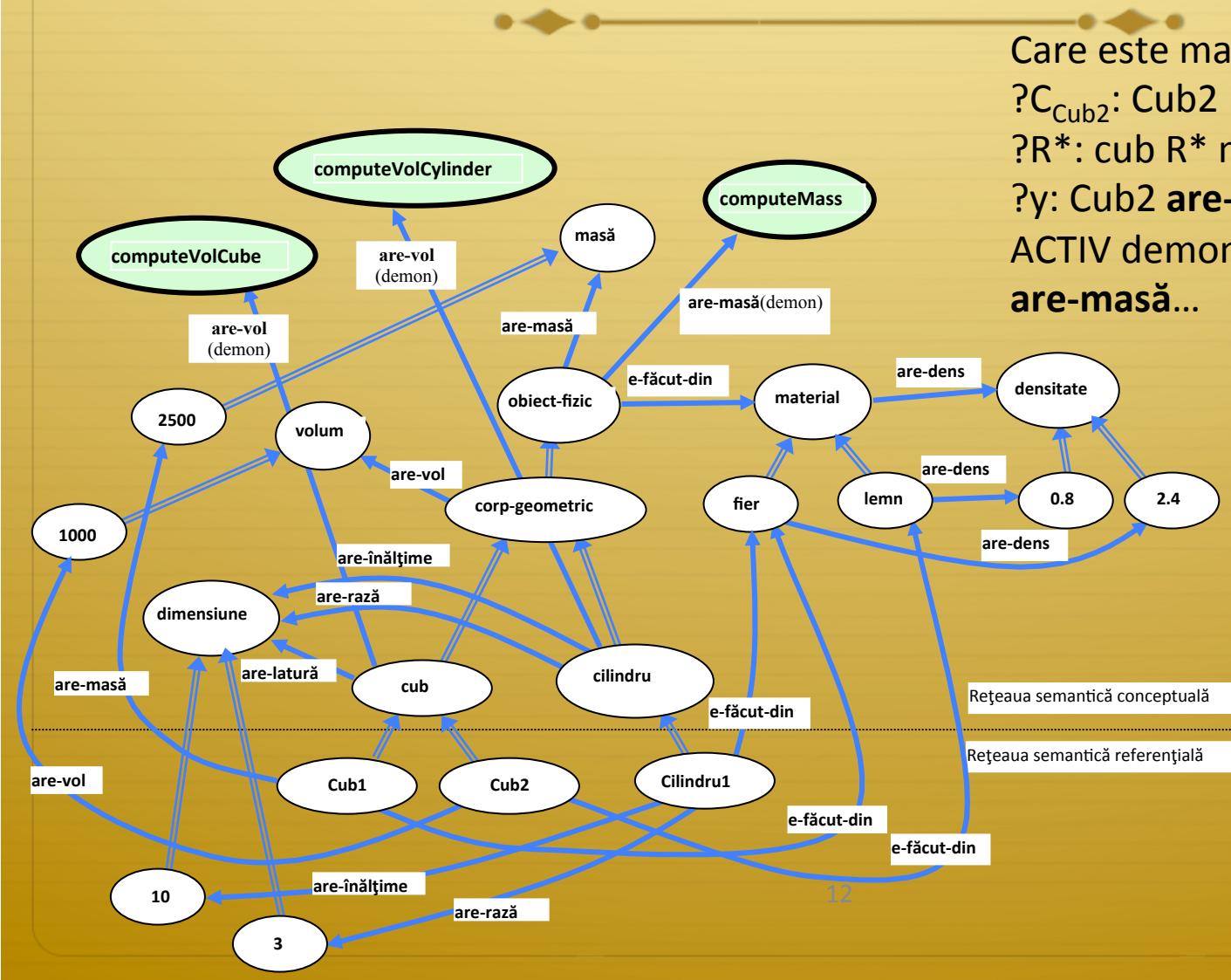
Care este masa lui Cub1?

?C: Cub1 ISA C → C = cub

?R*: cub R* masă → R* = are-masă

?y: Cub1 are-masă y → y = 2500

Demonul devine ACTIV



Care este masa lui Cub2?

? C_{Cub2} : Cub2 ISA $C_{Cub2} \rightarrow C_{Cub2} = \text{cub}$

? R^* : cub R* masă $\rightarrow R^* = \text{are-masă}$

?y: Cub2 are-masă y \rightarrow nil \rightarrow

ACTIV demonul din vârful relației
are-masă...

Rețeaua semantică conceptuală

Rețeaua semantică referențială

Demonul ComputeMass e activ!

```
procedure ComputeMass( x )
begin
; află densitatea lui x:
?Cx: x ISA Cx
?R1*: Cx R1* densitate
?y1: x R1* y1
; află volumul lui x:
?R2*: Cx R2* volum
?y2: x R2* y2
; calculează masa ca densitate
return y1 * y2;
end
```

cub2

$$m = \rho * V$$

$$\rightarrow C_x = \text{cub}$$

$$\rightarrow R_1^* = \text{e-făcut-din} \bullet \text{are-dens}$$

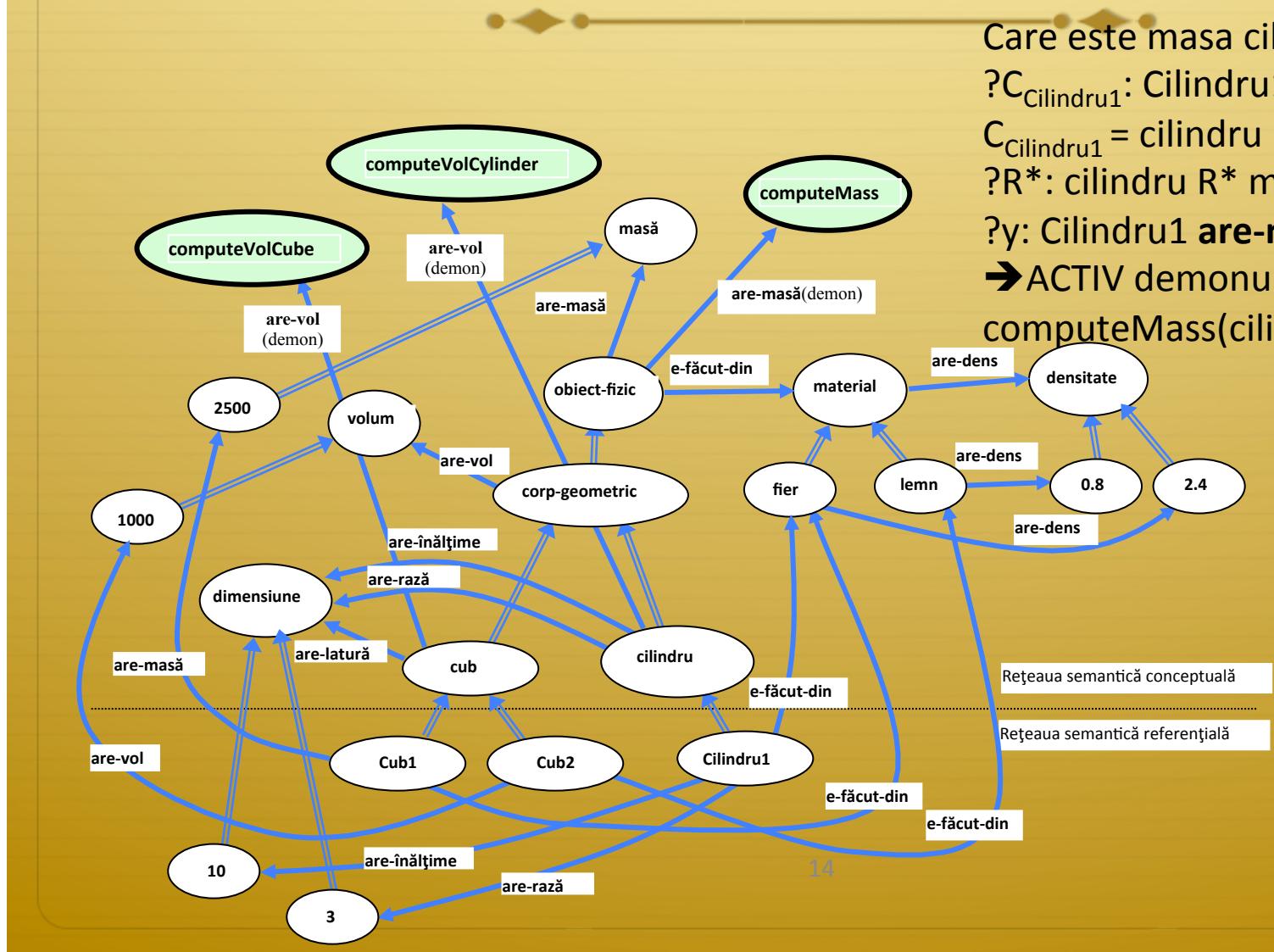
$$\rightarrow y_1 = \text{cub2 e-făcut-din} \bullet \text{are-dens} = 0.8$$

$$\rightarrow R_2^* = \text{are-vol}$$

$$\rightarrow y_2: \text{cub2 are-vol } y_2 \rightarrow y_2 = 1000$$

$$\text{return } 0.8 * 1000$$

Demoni într-o rețea semantică



Care este masa cilindrului 1?

?C_{Cilindru1}: Cilindru1 ISA C_{Cilindru1} →

C_{Cilindru1} = cilindru

?R*: cilindru R* masă → R* = are-masă

?y: Cilindru1 are-masă y → nil

→ ACTIV demonul
computeMass(cilindru1)

Rețeaua semantică conceptuală

Rețeaua semantică referențială

Demonul ComputeMass e activ!

```
procedure ComputeMass( x )
begin
; află densitatea lui x:
    ?Cx: x ISA Cx
    ?R1*: Cx R1* densitate
    ?y1: x R1* y1
; află volumul lui x:
    ?R2*: Cx R2* volum
    ?y2: x R2* y2
; calculează masa ca densitate * volum:
    return y1 * y2;
end
```

Cilindru1



$$m = \rho * V$$

→ C_x = cilindru

→ R₁* = e-făcut-din • are-dens

→ y₁ = Cilindru1 e-făcut-din • are-dens = 2.4

→ R₂*: Cilindru R₂* volum → R₂* = are-vol

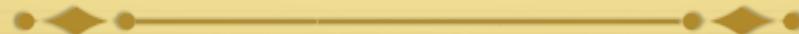
→ y₂: Cilindru1 are-vol y₂ → nil → ...

return 0.8 * 1000 = 800



Subiectul 3: Ontologii. Ce sunt OWL, Protégé?

Ontologies



- Rich conceptual schemas:
 - give formally defined meanings to the terms used in annotations, transforming them into semantic annotations
 - “Ontologies serve as metadata schemas, providing a controlled vocabulary of concepts, each with explicitly defined and machine-processable semantics. By defining shared and common domain theories, ontologies help people and machines to communicate concisely—supporting semantics exchange, not just syntax. Hence, the Semantic Web’s success and proliferation depends on quickly and cheaply constructing domain-specific ontologies.”

From: A. Maedche and S. Staab. Ontology learning for the semantic web.
IEEE Intelligent Systems, 16(2):72–79, 2001.

Ontology



- ❖ The study of ontology
 - ❖ traced back to the work of Plato and Aristotle
 - ❖ development of hierarchical categorisations of different kinds of entities and their distinguishing features:
 - ❖ the well known “tree of Porphyry” identifies animals and plants as sub-categories of living things distinguished by animals being sensitive, and plants being insensitive

Plasarea conceptului *bone* într-o ontologie

Class Hierarchy

Thing

- + [anatomical entity](#)
 - + [material anatomical entity](#)
 - + [anatomical structure](#)
 - + [multi-tissue structure](#)
 - + [bone](#)
 - + [endochondral bone](#)
 - [membrane bone](#)

Superclasses & Asserted Axioms

- [has_part](#) some [osteoblast](#)
- [part_of](#) some [skeletal system](#)
- [bone](#)

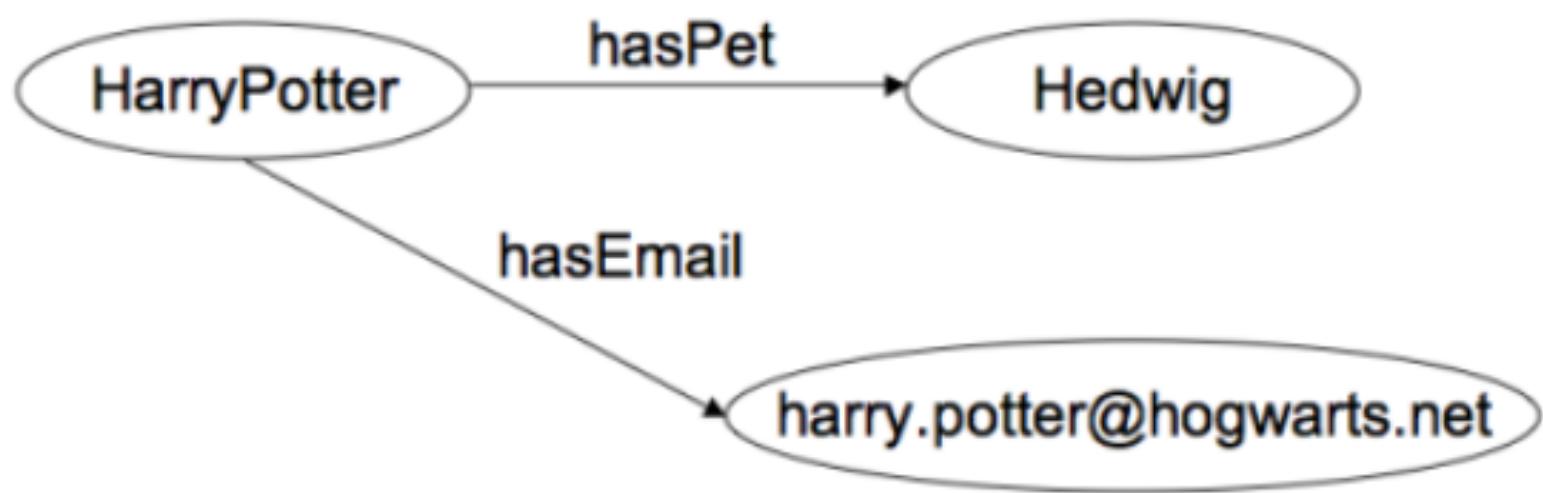
Class:skeletal system **Definition:** The rigid support system for the body.

http://www.ontobee.org/ontology/AEO?iri=http://purl.obolibrary.org/obo/AEO_0000085

OWL – un limbaj de descriere a ontologiilor

- ❖ The World Wide Web Consortium (W3C) set up a standardisation working group to develop a standard for a web ontology language => OWL ontology language standard
- ❖ OWL is based on Description Logics (DLs): a family of logic-based knowledge representation formalisms that are descendants of Semantic Networks and KL-ONE, but that have a formal semantics based on first order logic

Basic things in OWL



Concepts (classes)
Instances (individuals)
Properties (roles)

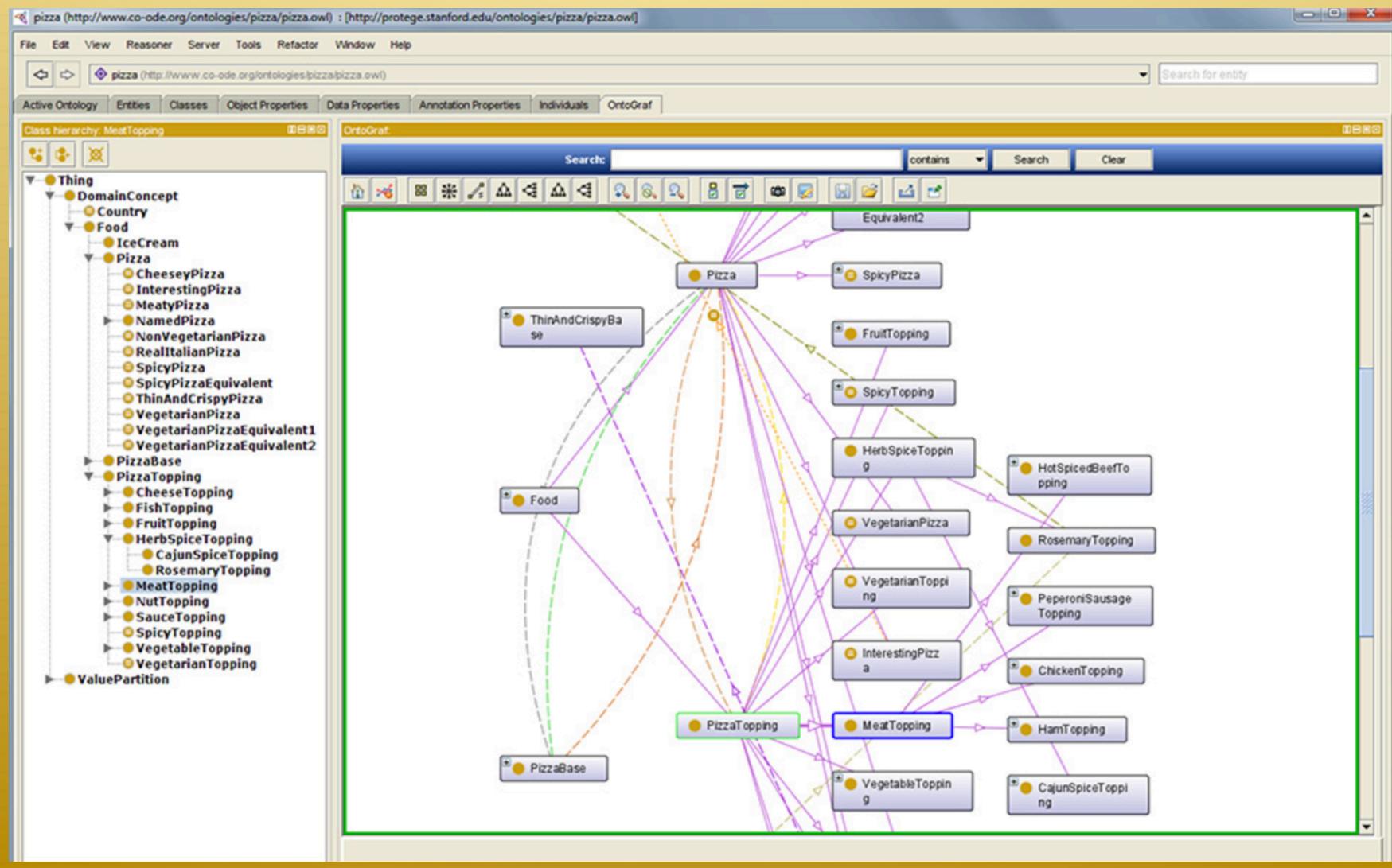
Protégé



- ❖ Mediu online de construire și editare de ontologii și creare de sisteme inteligente
- ❖ *Protégé Desktop supports creation and editing of one or more ontologies in a single workspace via a completely customizable user interface. Visualization tools allow for interactive navigation of ontology relationships. Advanced explanation support aids in tracking down inconsistencies. Refactor operations available including ontology merging, moving axioms between ontologies, rename of multiple entities, and more.*

<https://protege.stanford.edu/>

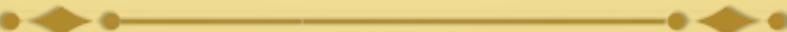
Protégé



Subiectul 4:

Web-ul semantic. Dați un exemplu de interogare care nu poate fi rezolvată de un motor de căutare precum Google, dar ar putea fi rezolvată prin raționament în web-ul semantic. Sugerați o soluție la întrebare.

Semantic Web



- ❖ The goal of semantic web research: to allow the vast range of web-accessible information and services to be more effectively exploited by both humans and automated tools.
- ❖ Exploitation of the vast web
 - ❖ through RDF and OWL: standard formats for the sharing and integration of data and knowledge—the latter in the form of rich conceptual schemas called **ontologies**

Examples of queries that cannot be answered by the actual web search engines



- ❖ The list of presidents of EU countries
 - ❖ List of EU countries:
 - ❖ https://europa.eu/european-union/about-eu/countries/member-countries_en
 - ❖ List of presidents of countries:
 - ❖ https://en.wikipedia.org/wiki/List_of_current_heads_of_state_and_government

Examples of queries...



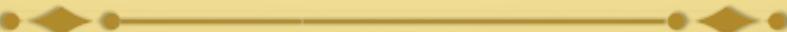
- ❖ German writers contemporary with Beethoven
 - ❖ a book: **Music and Literature in German Romanticism**, by Siobhán Donovan and Robin Elliott
 - ❖ abstract, presented at
<http://www.jstor.org/stable/10.7722/j.ctt81t1f>
 - ❖ an Wikipedia article: Beethoven and his contemporaries
 - ❖ but mainly composers, at
https://en.wikipedia.org/wiki/Beethoven_and_his_contemporaries

German writers contemporary with Beethoven



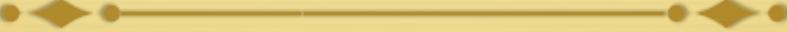
- ❖ One possible solution:
 - ❖ Beethoven has lived between T1 and T2
 - ❖ German writers borned between T1-20 and T2-20

Examples of queries...

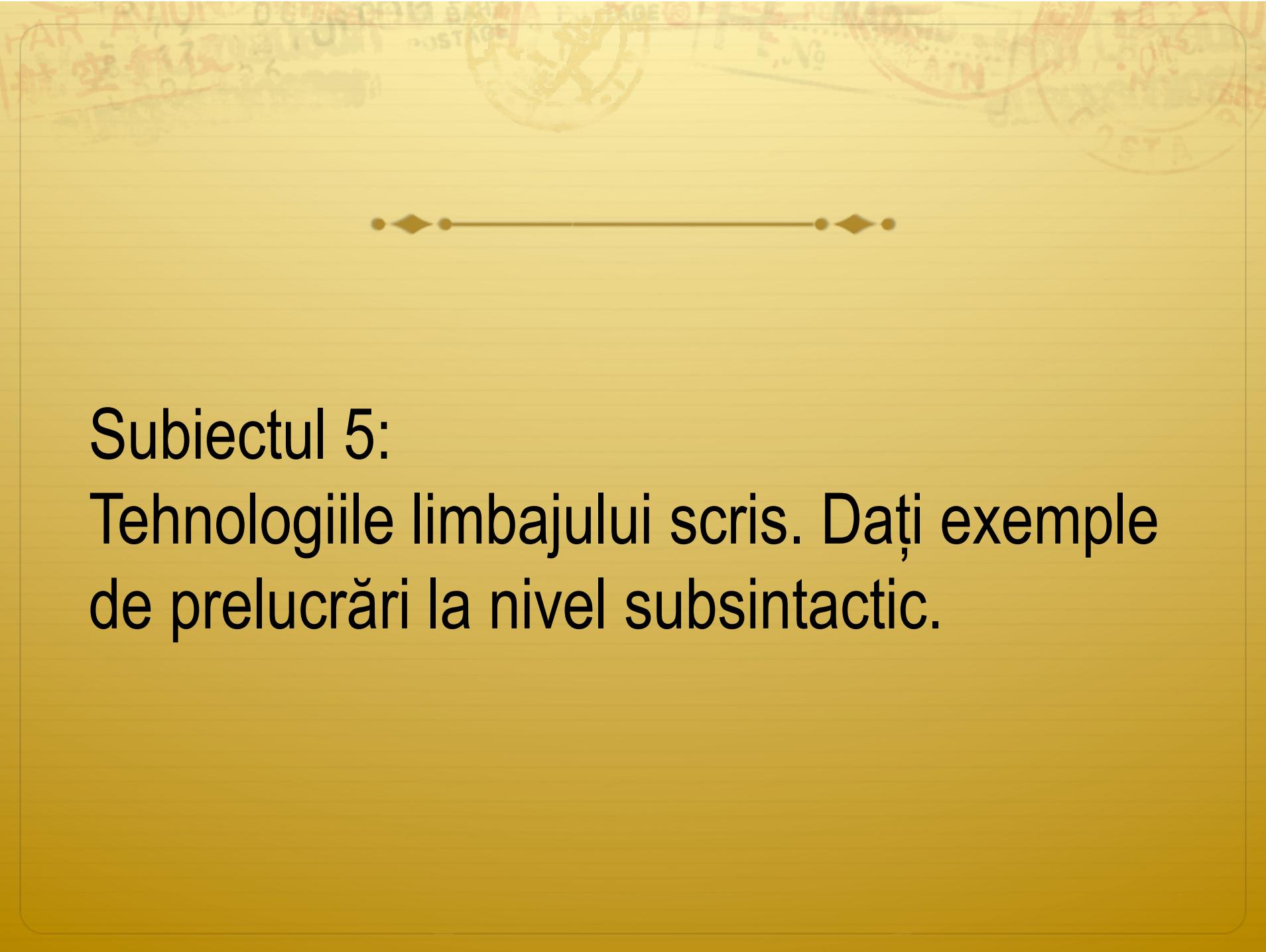


- Classical example of a semantic web application:
 - an automated travel agent that, given various constraints and preferences, would offer the user suitable travel or vacation suggestions
 - key feature of such a “software agent”: it would not simply exploit a predetermined set of information sources, but would search the web for relevant information in much the same way that a human user might do when planning a vacation

Key idea behind the semantic web



- ❖ Address the problem of offering automated reasoning by giving the machine accessible semantics to annotations.
- ❖ Achieved through **ontologies**
- ❖ Areas:
 - ❖ knowledge representation and reasoning, databases, computational linguistics, computer vision, agent systems

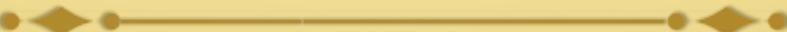


Subiectul 5:
Tehnologiile limbajului scris. Dați exemple
de prelucrări la nivel subsintactic.

Tehnologiile limbajului scris

- 
- ❖ Analiza și înțelegerea limbajului
 - ❖ prelucrări sub-sintactice
 - ❖ unitățile lexicale
 - ❖ granițele de frază
 - ❖ granițele de propoziții
 - ❖ partea de vorbire și marca morfologică
 - ❖ lema
 - ❖ numele de entități
 - ❖ grupurile (nominale, verbale, prepoziționale etc.) și atracțiile lexicale (colocații)

Instrumente de bază în PLN



- ❖ **Tokenizer:** determină granițele unităților lexicale
 - ❖ intrare: text (șir de caractere)
 - ❖ ieșire: <tok id="...">cuvânt</tok>
 - ❖ cum: prin expresii regulate

Instrumente de bază în PLN

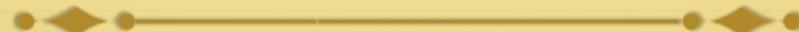


- ❖ **POS-Tagger:** etichetare la parte de vorbire (dezambiguizare morfosintactică)
 - ❖ intrare: <tok id="...">cuvânt</tok>
 - ❖ ieșire: <tok id="..." POS="...>cuvânt</tok>
 - ❖ cum: exploatând frecvențele de apariție a anumitor secvențe de părți de vorbire => optimizare globală a secvențelor de etichete

The saw made noise.



Instrumente de bază în PLN

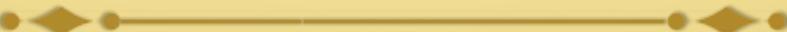


- ❖ **Lematizator:** determină forma de bază a cuvintelor
 - ❖ intrare: <tok id="..." POS="...">word</tok>
 - ❖ ieșire: <tok id="..." POS="..." lemma="...">word</tok>
 - ❖ cum: pe baza unui dicționar de leme și exploatând frecvențe de apariție a secvențelor de leme => optimizare globală

The saw made noise.

the saw made noise
see see make noise

Instrumente de bază în PLN



- ❖ **NP-Chunker:** detectează grupuri nominale
 - ❖ intrare: secvențe de elemente <tok>
 - ❖ ieșire: <np id="...">...</np>
 - ❖ cum: aplicând expresii regulate

Instrumente de bază în PLN



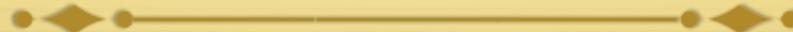
- ❖ **NER (name entity recogniser)**: recunoaște și clasifică nume de entități
 - ❖ intrare: text
 - ❖ ieșire: <ne id="..." type ="...>...</ne>
 - ❖ cum: pe bază de expresii regulate și liste foarte mari de nume de entități specializează pe limbi (gazeteers)

Subiectul 6:

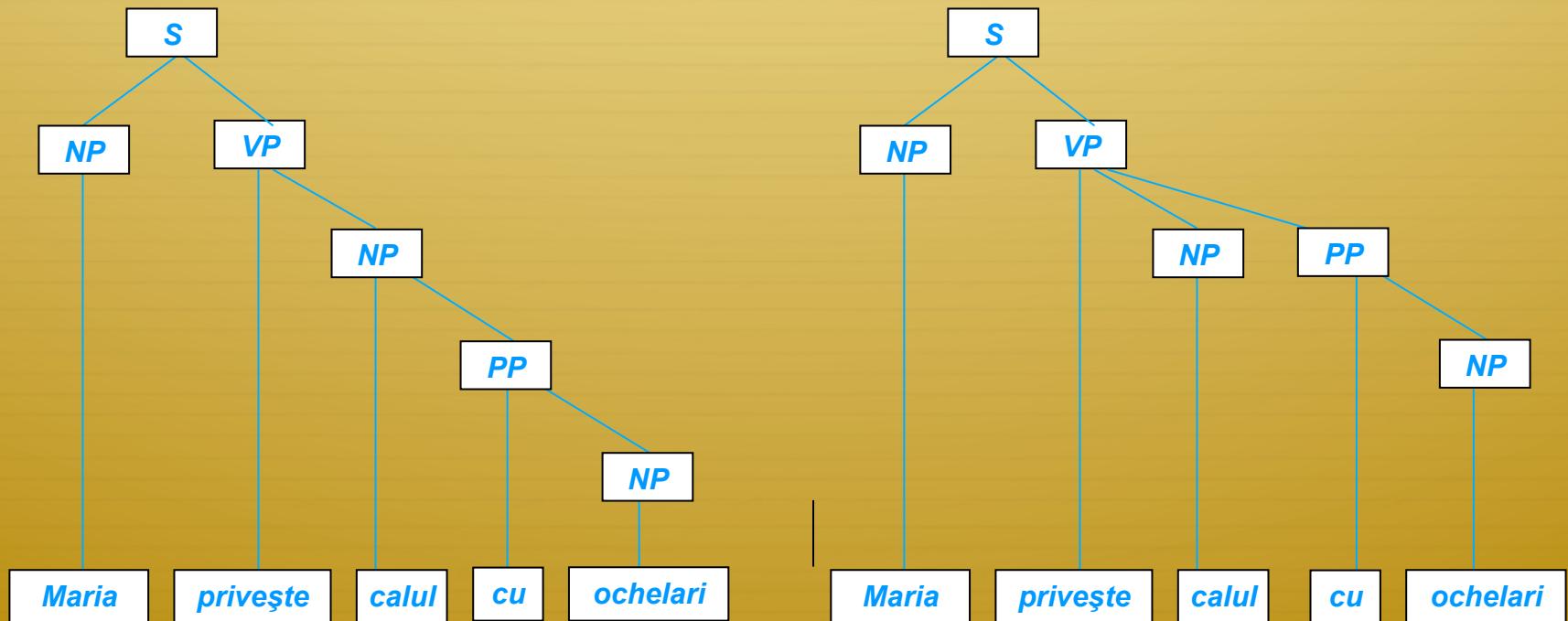
Nivelul sintactic în analiza limbajului scris.

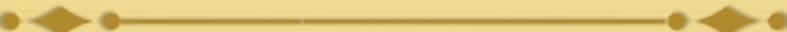
Dați un exemplu de propoziție ambiguă
sintactic și desenați arborii de analiză.

Ambiguități sintactice



Maria privește calul cu ochelari.

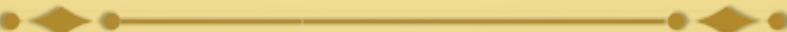




Subiectul 7:

Puneți în evidență relații anaforice și semantice în textul următor: “Vreme de patruzeci de ani viața Ellei Rubinstein fusese ca o apă stătătoare... Sotul ei, David, era un dentist de succes...”

Relații de rudenie: exemplu



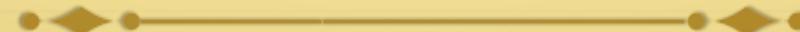
- Vreme de patruzeci de ani viața *Ellei Rubinstein*,₁ fusese ca o apă stătătoare... *Sotul ei*,₁ *David*, era un dentist de succes...

Apoziție: Rel Per-X_{pron,gen}, Per-Y, =>

marriage(antecedent(X):person[sex:?:], Y:person[sex:?:])

marriage(Ella Rubistein:person[sex:f],
David:person[sex:m])

Relații anaforice



- *coref*
- *coref-interpret*
- *member-of, has-as-member* (inverse)
- *isa, class-of* (inverse)
- *part-of, has-as-part* (inverse)
- *subgroup-of, has-as-subgroup* (inverse)
- *has-name, name-of* (inverse)



1:[Acteea]... 2:[tânăra libertă]... => [2] coref [1]

1:[mâna 2:[lui] dreaptă] => [1] part-of [2]

Relații de rudenie

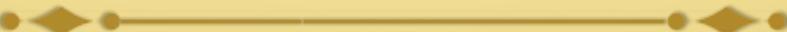


- *parent-of*
- *child-of*(inverse of *parent-of*)
- *grandparent-of* and *grandchild-of*(inverse)
- *sibling* (symmetrical)
- *ant-uncle-of*, *nephew-of*(inverse relation)
- *cousin-of*(symmetrical)
- *spouse-of*(symmetrical)
- *unknown*



1:[*celui de-al doilea soț* 2:[*al Popeii*]] => [1] spouse-of [2]

1:[*sora lui* 2:[*Petronius*]] => [1] sibling-of [2]

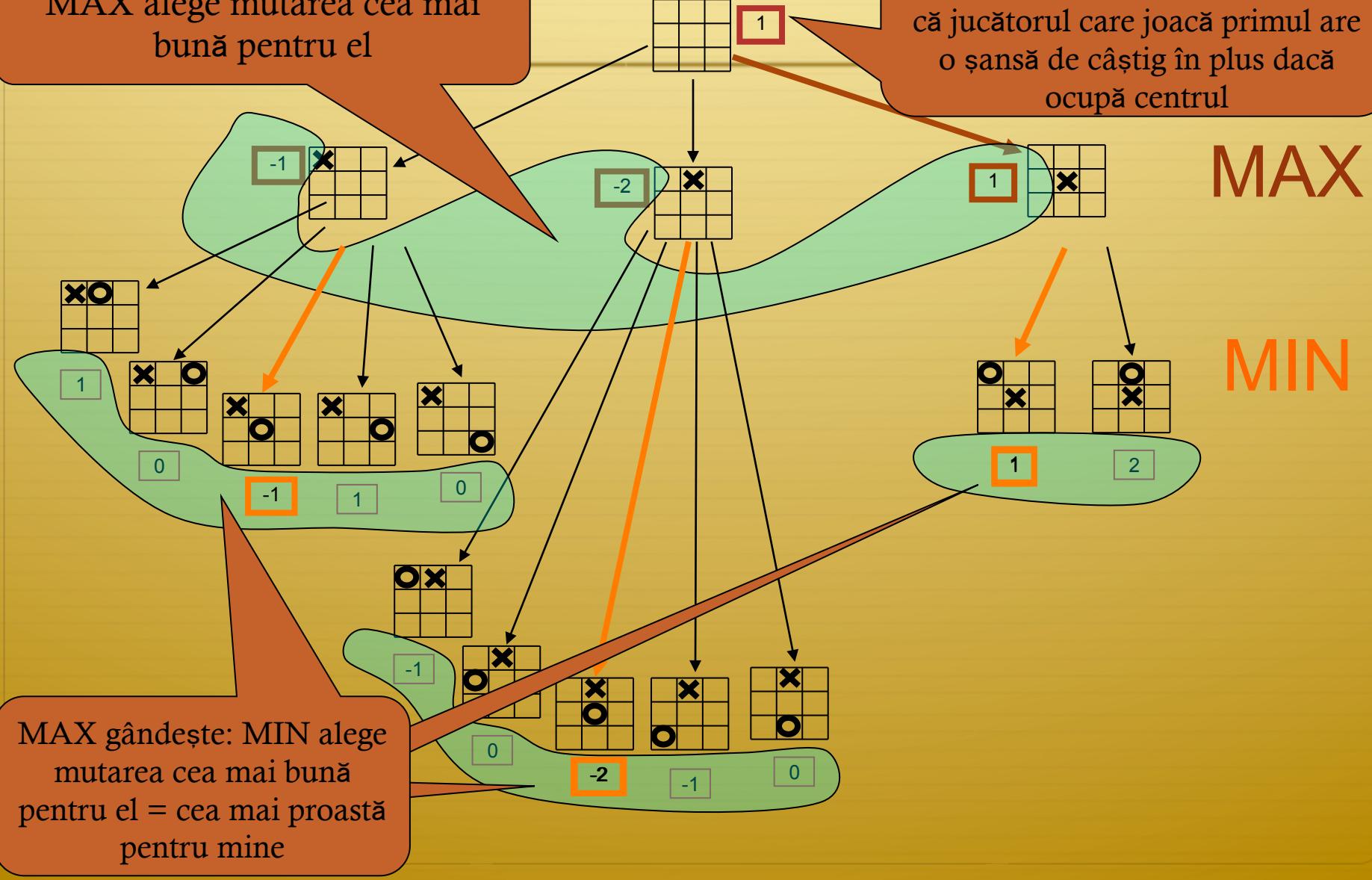


Subiectul 8: Jocuri. Algoritmul MIN-MAX. Simplificarea ALPHA-BETA.

Evaluarea: de jos în sus

MAX alege mutarea cea mai bună pentru el

O dezvoltare a spațiului de joc pe o adâncime de 2 duce la concluzia că jucătorul care joacă primul are o sansă de câștig în plus dacă ocupă centrul



Metoda MIN-MAX

```
function min-max(state, player, depth)
begin
    if (depth = 0) then return score(state);
    val = worst(player);
    while (mai sunt stări de generat) begin
        generez o stare -> s;
        val <- back-up-compare(val, min-max(s, not(player), depth-1), player);
            // următoarea mișcare micșorează spațiul de căutare în cazul în care se obține
            // poziția de câștig într-una
            // din stările generate:
        if (val = -worst(player)) return(val);
    end
    return(val);
end

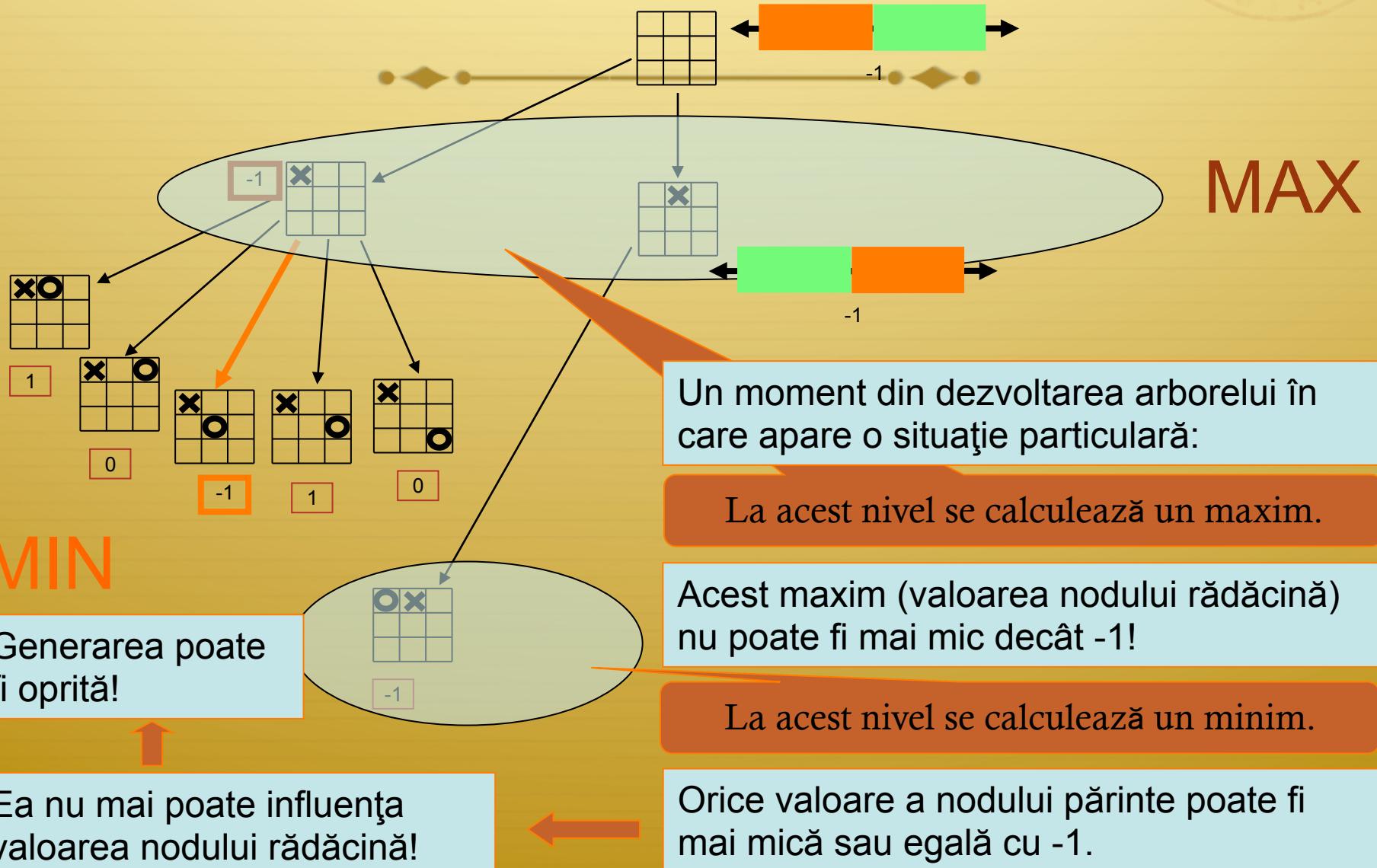
function worst(player)
begin
    if player = MAX then return -∞;
    else return +∞;
end

function back-up-compare(val1, val2, player)
begin
    if player = MAX then return max(val1, val2);
    else return min(val1, val2);
end
```

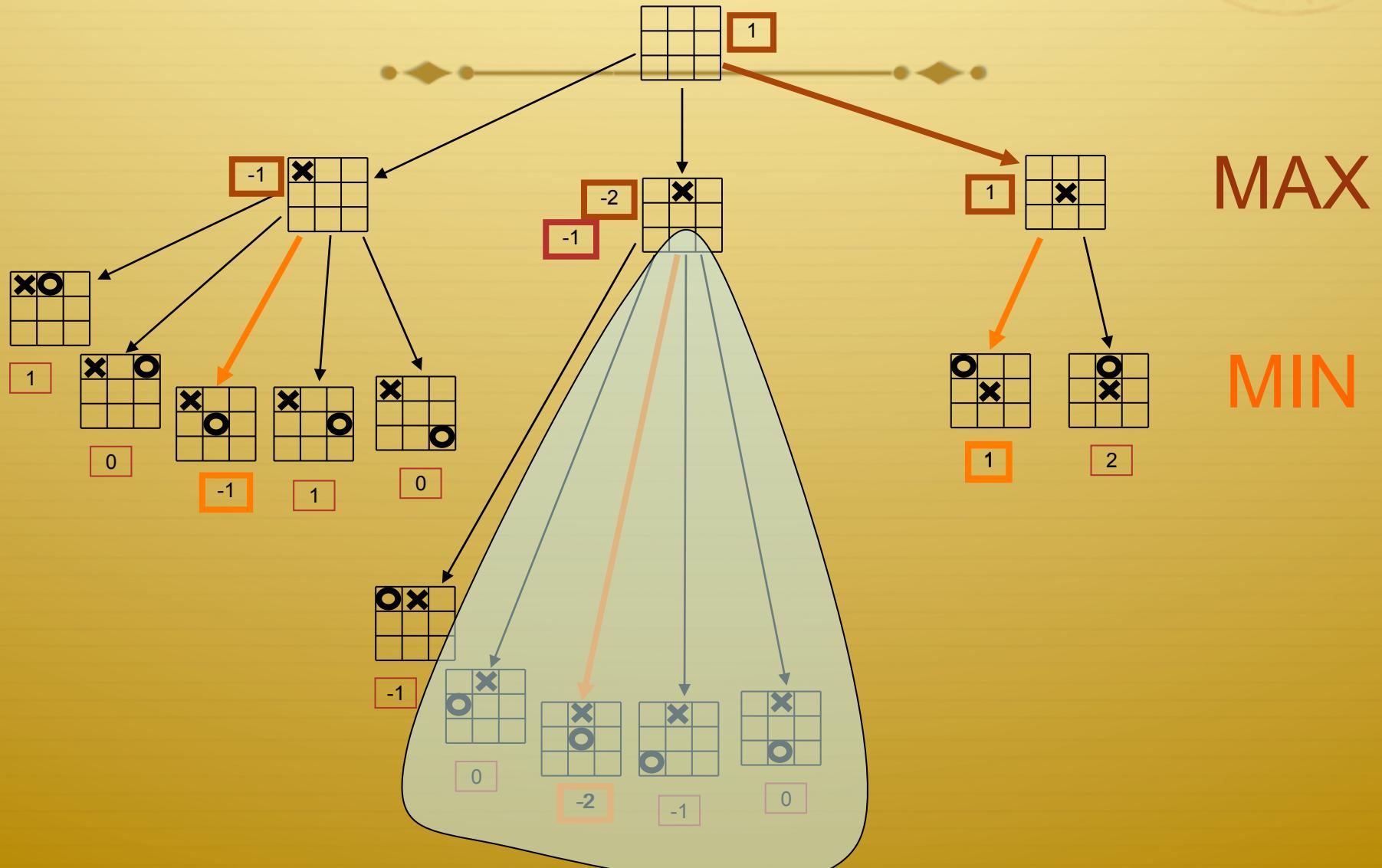
Apelul:

min-max( ,MAX,2)

Metoda alpha-beta



Metoda alpha-beta



Metoda alpha-beta

```
function alpha-beta(state, player, depth)
begin
    if (depth = 0) then return score(state);
    val = worst(player);
    while (mai sunt stări de generat) begin
        generez o stare -> s;
        newval <- alpha-beta(s, not(player), depth-1);
        if player=MAX & newval ≤ val then return(newval);
        else if player=MIN & newval ≥ val then return(newval);
        else val ← back-up-compare(val, min-max(s, not(player)), depth-1), player);
        // următoarea mișcare micșorează spațiul de căutare în cazul în care se
        obține poziția de câștig
        // într-una din stările generate:
        if (val = -worst(player)) return(val);
    end
    return(val);
end

function worst(player)
begin
    if player = MAX then return -∞;
    else return +∞;
end

function back-up-compare(val1, val2, player)
begin
    if player = MAX then return max(val1, val2);
    else return min(val1, val2);
end
```

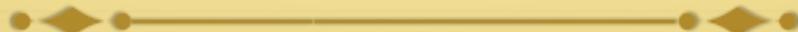
Apelul:

alpha-beta (■■■,MAX,2)



Subiectul 9:
Rezolvarea problemelor de IA. Stări și
reprezentarea lor, spațiul stărilor, tranzitii
și reprezentarea lor, strategii de căutare a
soluției.

Cele 5 cerințe în modelarea unei probleme de IA

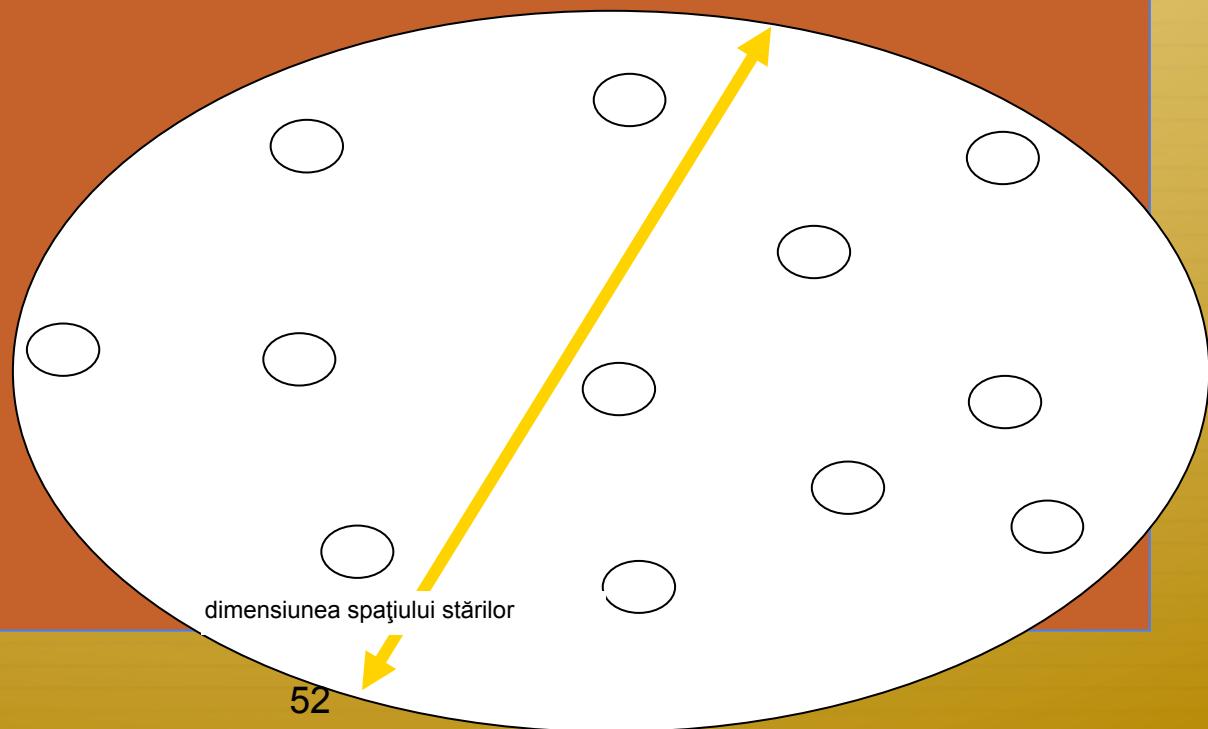


- Pasul 1** Diferențiază problema generală de instanțele ei
- Pasul 2** Recunoaște o stare și apreciază dimensiunea spațiului stărilor
- Pasul 3** Găsește cea mai adecvată reprezentare a stărilor
- Pasul 4** Reprezintă tranzițiile dintre stări
- Pasul 5** Alege o strategie de control

Spațiul problemei



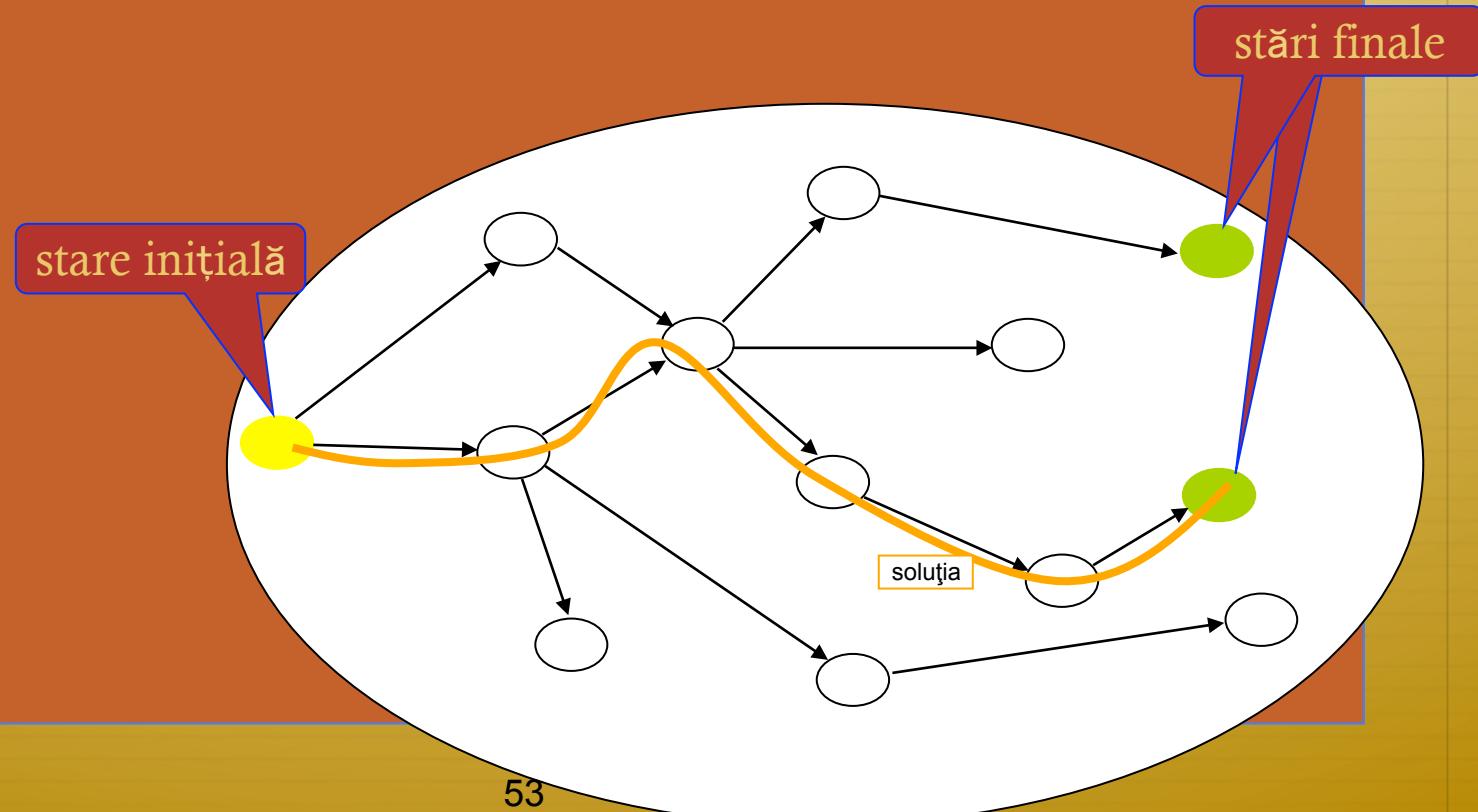
- ❖ Stări, dimensiunea spațiului



Stări, spațiul stărilor, dimensiunea lui



Soluția = un sir de tranzitii



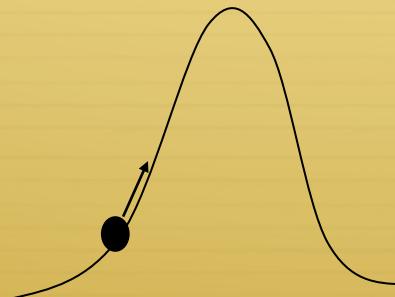
Căutarea soluției



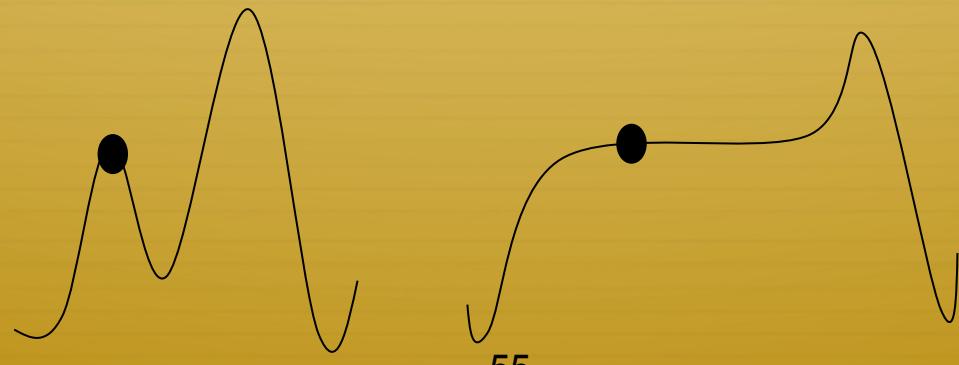
- ❖ Algoritmi și euristici de căutare în spațiul stăriilor
 - ❖ Strategii irevocabile
 - ❖ ascensională (*hill-climbing*)
 - ❖ Strategii tentative
 - ❖ ascensională cu revenire (*backtracking*)
 - ❖ Strategii exhaustive (*brute-force*)
 - ❖ generează-și-testează
 - ❖ Întâi-în-adâncime (*depth-first*)
 - ❖ Întâi-în-lărgime (*breadth-first*)
 - ❖ cel mai bun întâi (*best-first*)

Strategii irevocabile: *hill-climbing*

- ❖ Cale de întoarcere nu există
 - ❖ o funcție apreciază apropierea de soluție



- ❖ pericole: maxime locale, platouri

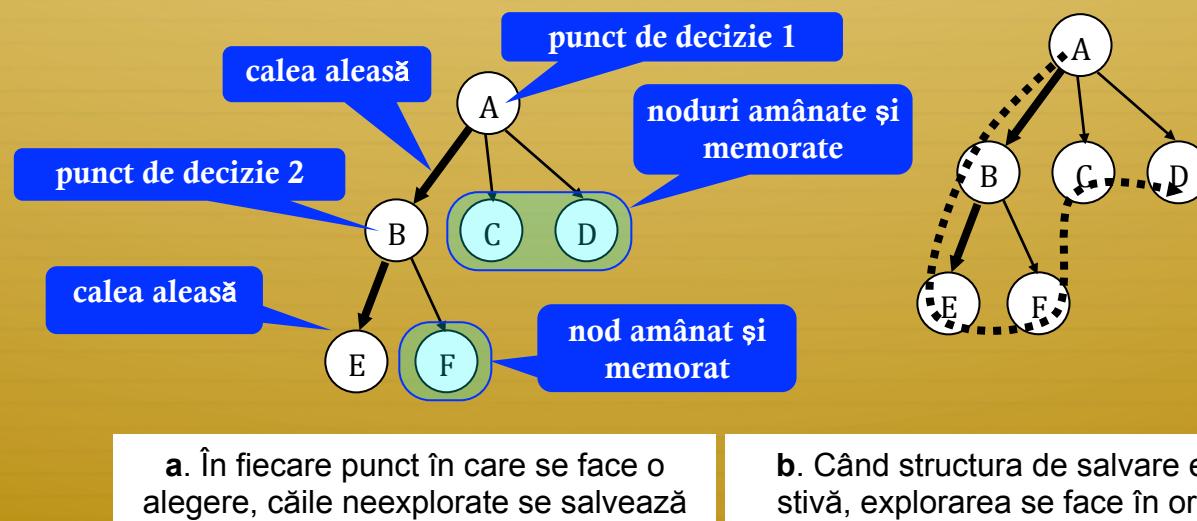


Hill climbing

```
procedure hill-climbing(initial-state)
begin
    current-state <- initial-state;
    while(current-state) {
        if (current-state e stare finală) return current-state;
        all-new-neighbour-states <- setul stărilor ce pot fi obținute din
current-state prin operatorii aplicabili ei;
        all-new-neighbour-states <- all-new-neighbour-states minus toate
stările deja vizitate;
        sortează all-new-neighbour-states în ordinea descrescătoare a
valorilor funcției de cost;
        all-new-neighbour-states <- all-new-neighbour-states minus stările
de valoare mai mică decât a lui current-state;
        if (all-new-neighbour-states ≠ ∅) current-state <- prima clasată
în all-new-neighbour-states);
        else return FAIL;
    }
    return fail;
end
```

Strategii tentative: *backtracking*

- ❖ Dacă o stare nu mai are succesișori "iau urma îndărăț"
- ❖ o memorie în care se plasează la fiecare pas stările vecine, diferite de cea în care se efectuează tranziția



Backtracking hill-climbing

Pasul 5

```
procedure backtracking-hill-climbing(initial-state)
begin
    heap <- initial-state ° Ø;
    solution <- Ø;
    while(heap) {
        current-state <- first(stack);
        heap <- rest(heap);
        solution <- solution ° current-state;
        if (current-state e stare finală) return solution;
        all-new-neighbour-states <- setul stărilor ce pot fi
        obținute din current-state prin operatorii aplicabili ei;
        all-new-neighbour-states <- all-new-neighbour-states \
        toate stăriile deja vizitate;
        heap <- heap ° all-new-neighbour-states;
        sort heap descendent după valorile funcției cost;
        heap <- heap \ stăriile de valori mai mici decât a lui
        current-state;
    }
    return FAIL;
end
```

Metode de căutare sistematică (*brute-force*)

Căutare întâi-în-adâncime (depth-first search – DFS)

memoria: stivă

```
function depthFirstSearch(root)
begin
    stack <- push(root, ∅); solution <- ∅;
    while (stack not empty)
        { node <- pop(stack);
          solution <- solution ° node;
          if goal(node) then return solution;
          else push(node's successors, stack);
        }
    return FAIL;
end
```

Metode de căutare sistematică (*brute-force*)

- ❖ Căutare întâi-în-lărgime (*breadth-first search – BFS*)

- ❖ memoria: coadă

```
function breadthFirstSearch(root)
begin
    queue <- in(root, ∅); solution <- ∅;
    while (queue not empty)
        { node <- out(queue);
          solution <- solution ° node;
          if goal(node) then return node;
          else in(node's successors, queue);
        }
    return FAIL;
end
```

Metode de căutare sistematică (*brute-force*)

- ❖ Căutare cel-mai bun-întâi (*best-first search*)

- ❖ memoria: listă; o funcție euristică de cost
function bestFirstSearch(root)

```
begin
```

```
    list <- include(root, ∅); solution <- ∅;
```

```
    while (list not empty)
```

```
        { node <- get-first(list);
```

```
            solution <- solution ° node;
```

```
            if goal(node) then return solution;
```

```
            else
```

```
                { include(node's successors, list);
```

```
                    sort list descending;
```

```
                }
```

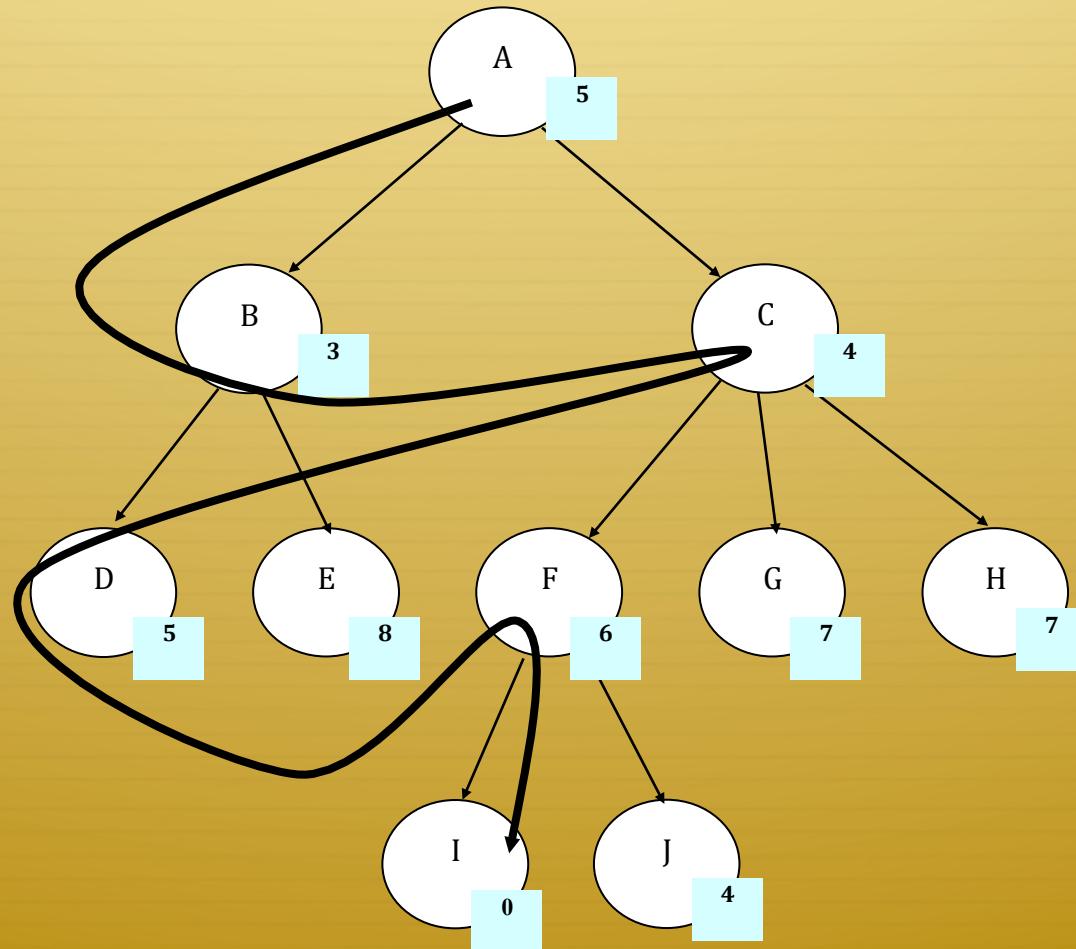
```
}
```

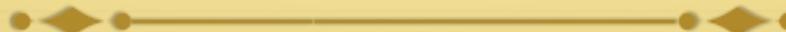
```
    return FAIL;
```

```
end
```

Exemplu: best-first search

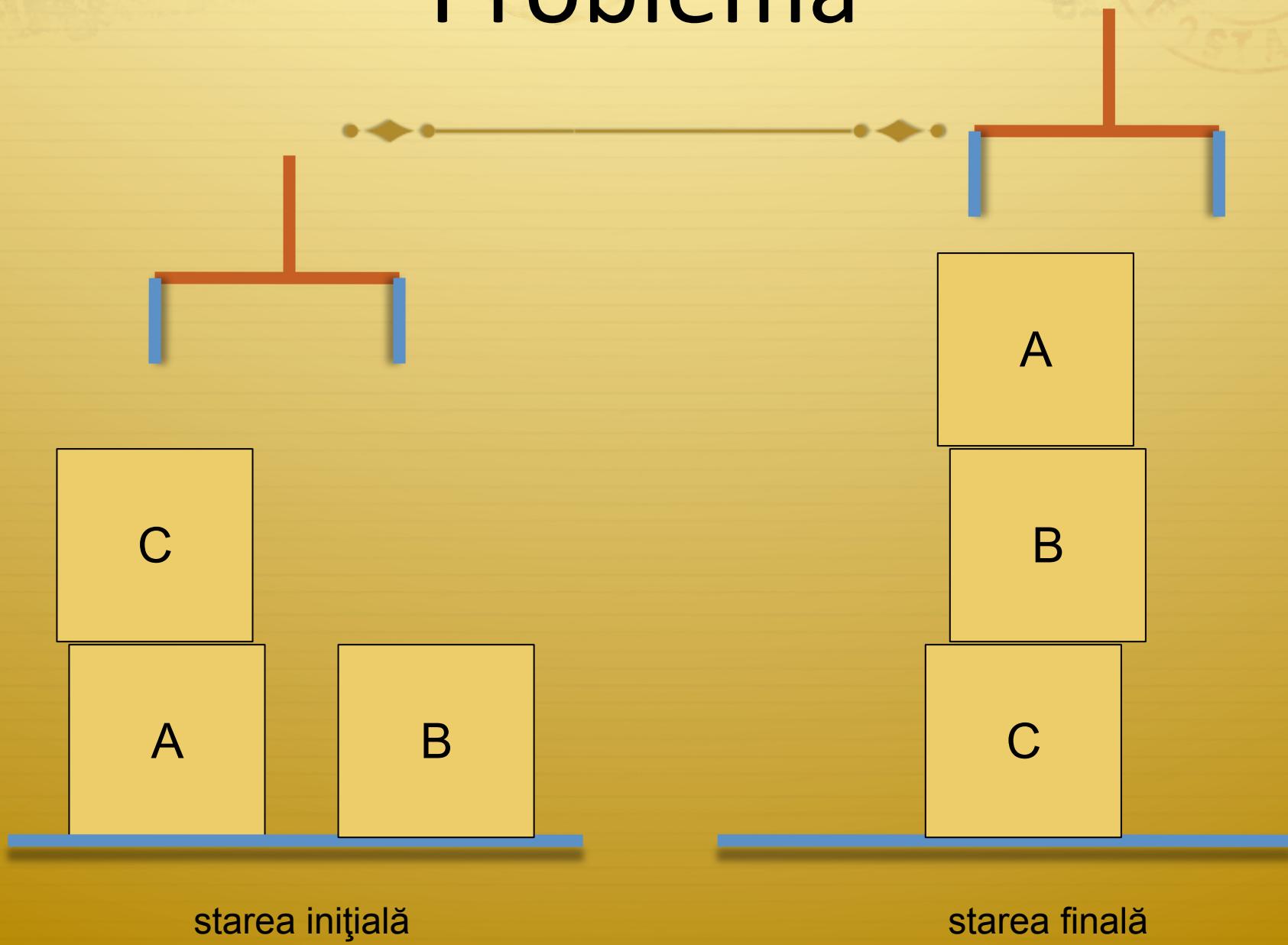
Pasul 5



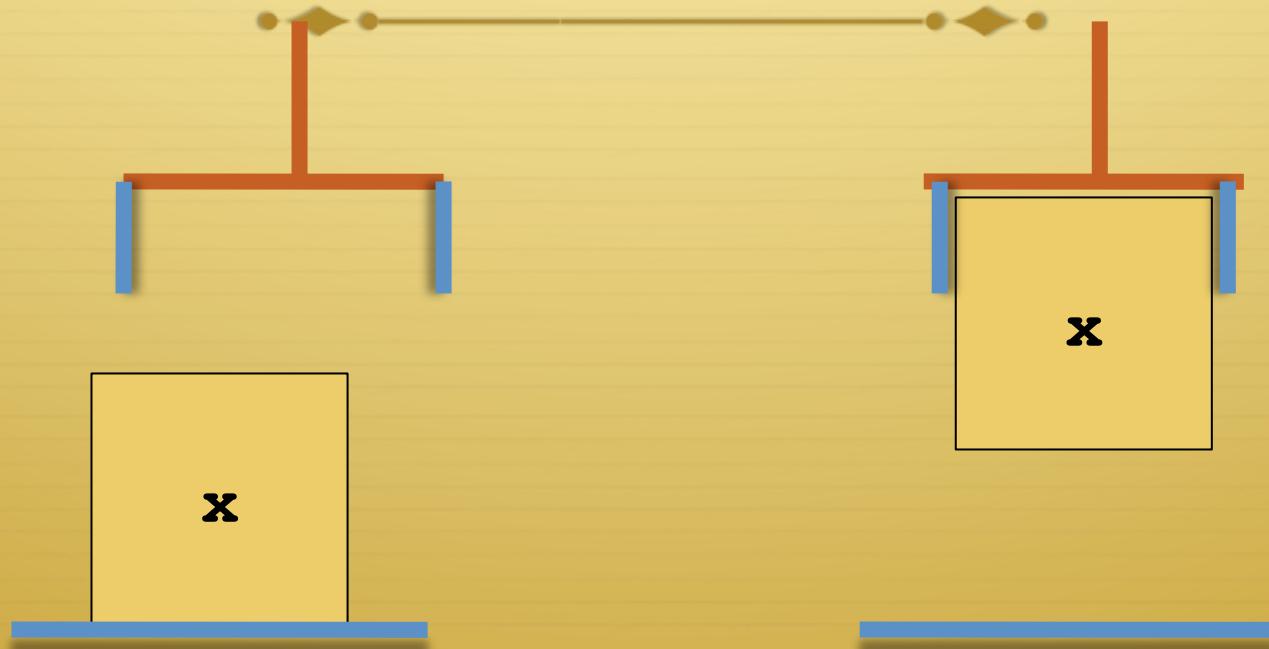


Subiectul 10:
Planificare și execuția planurilor, reguli
STRIPS, diagrama triunghiulară,
revenirea în plan în caz de accident.

Problema



Reguli STRIPS

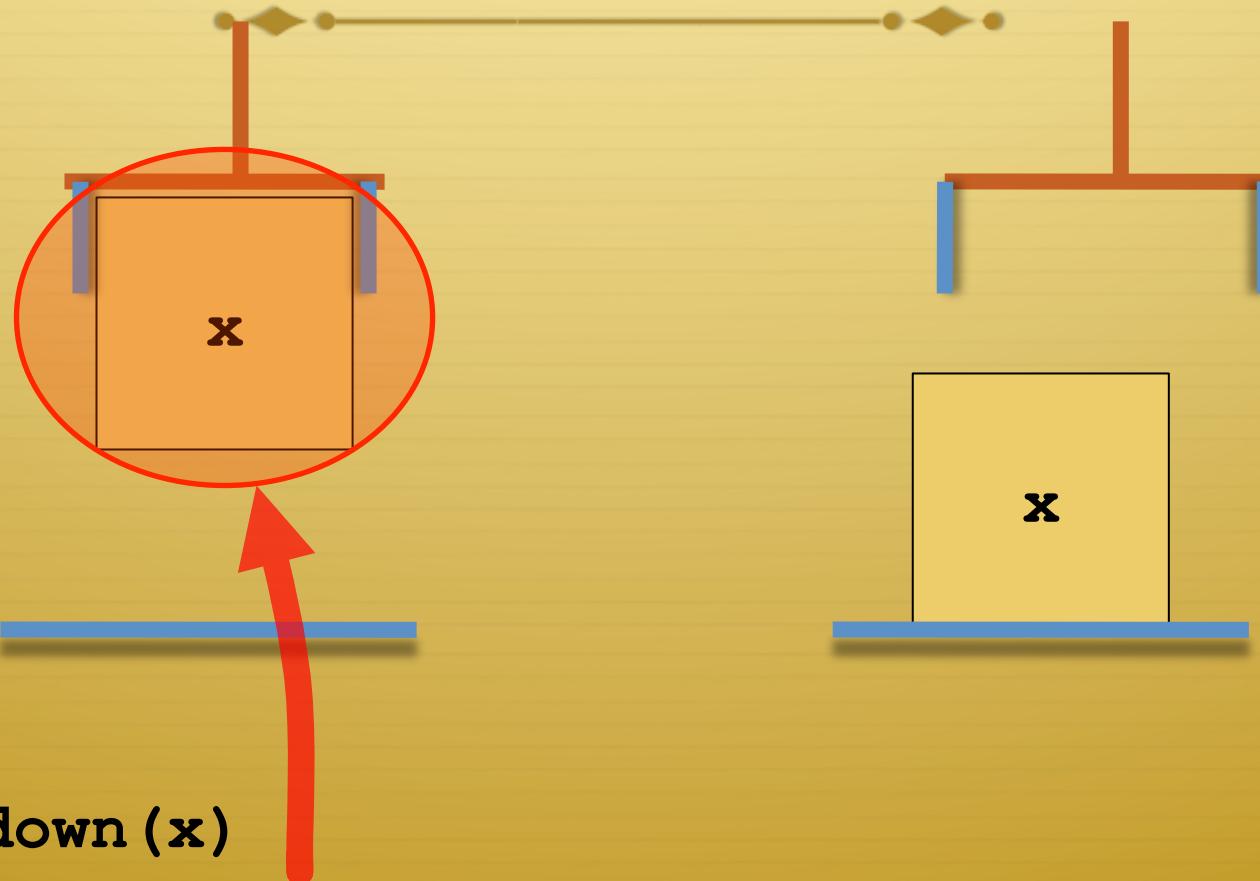


`pickup(x)`

P&D: `ontable(x)`, `clear(x)`, `handempty`

A: `holding(x)`

Reguli STRIPS

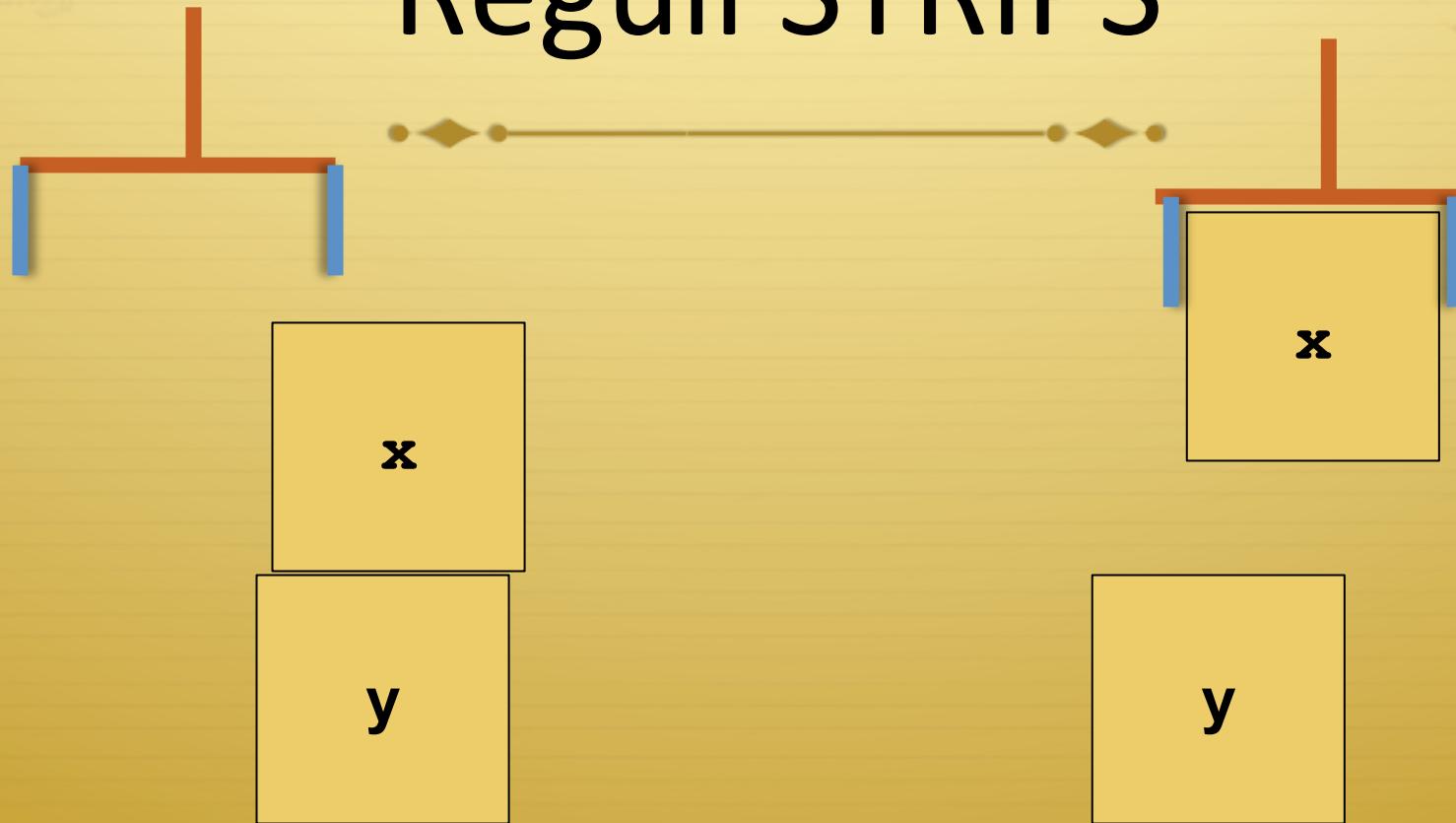


`putdown (x)`

P&D: **holding (x)**

A: `ontable(x) , clear(x) , handempty`

Reguli STRIPS

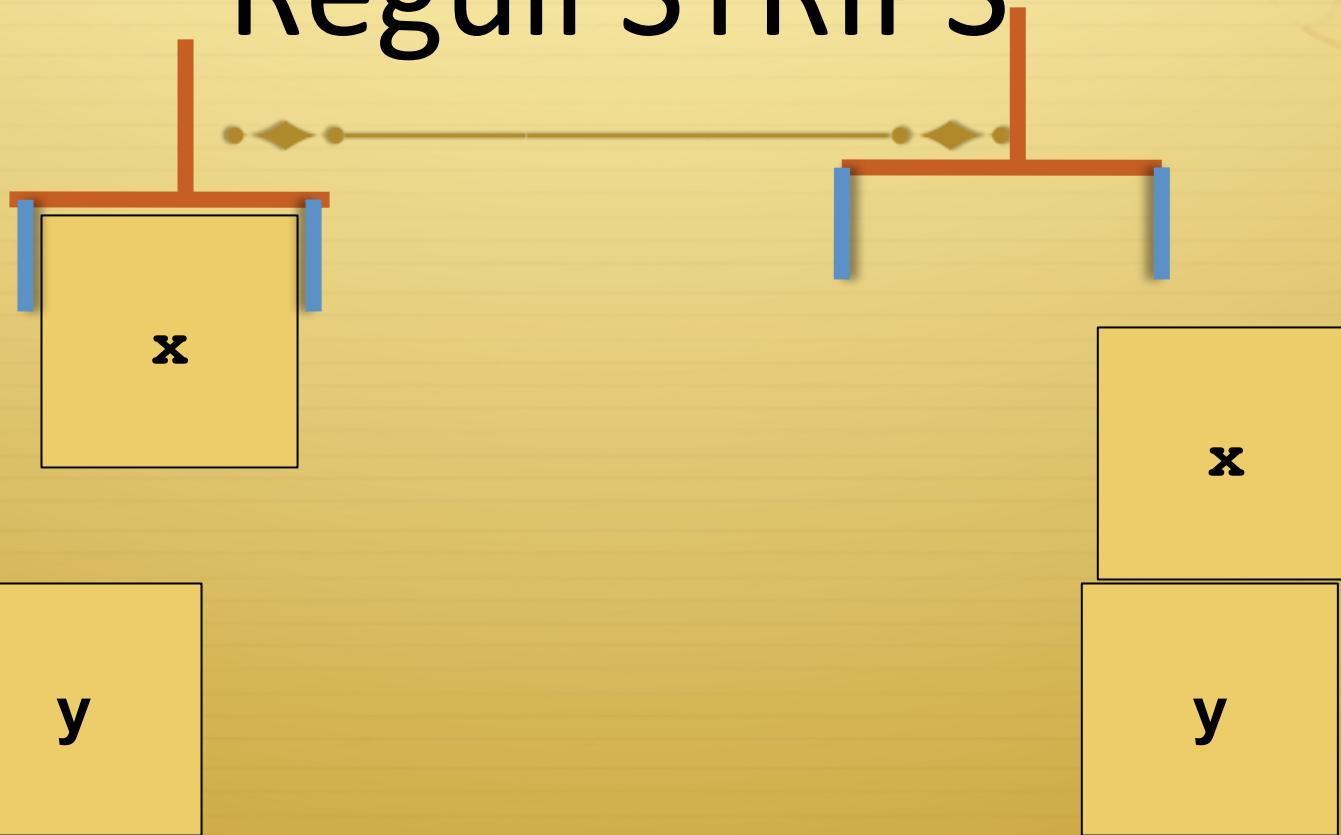


unstack(x,y)

P&D: `on(x,y)`, `clear(x)`, `handempty`

A: `holding(x)`, `clear(y)`

Reguli STRIPS



`stack(x,y)`

P&D: `holding(x)`, `clear(y)`

A: `on(x,y)`, `handempty`, `clear(x)`

Diagrama triunghiulară

$N = \text{lungimea soluției în #pași}$

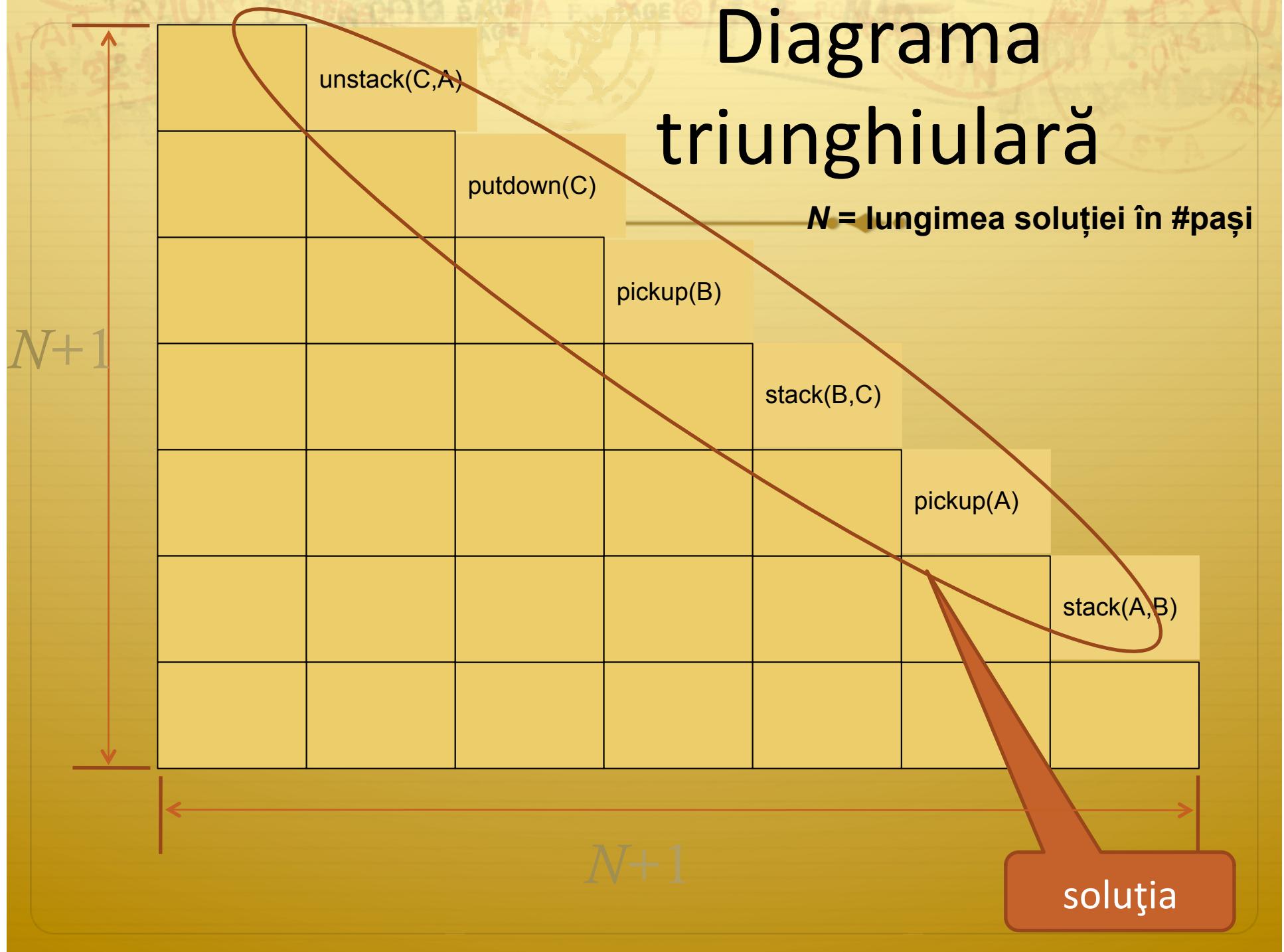
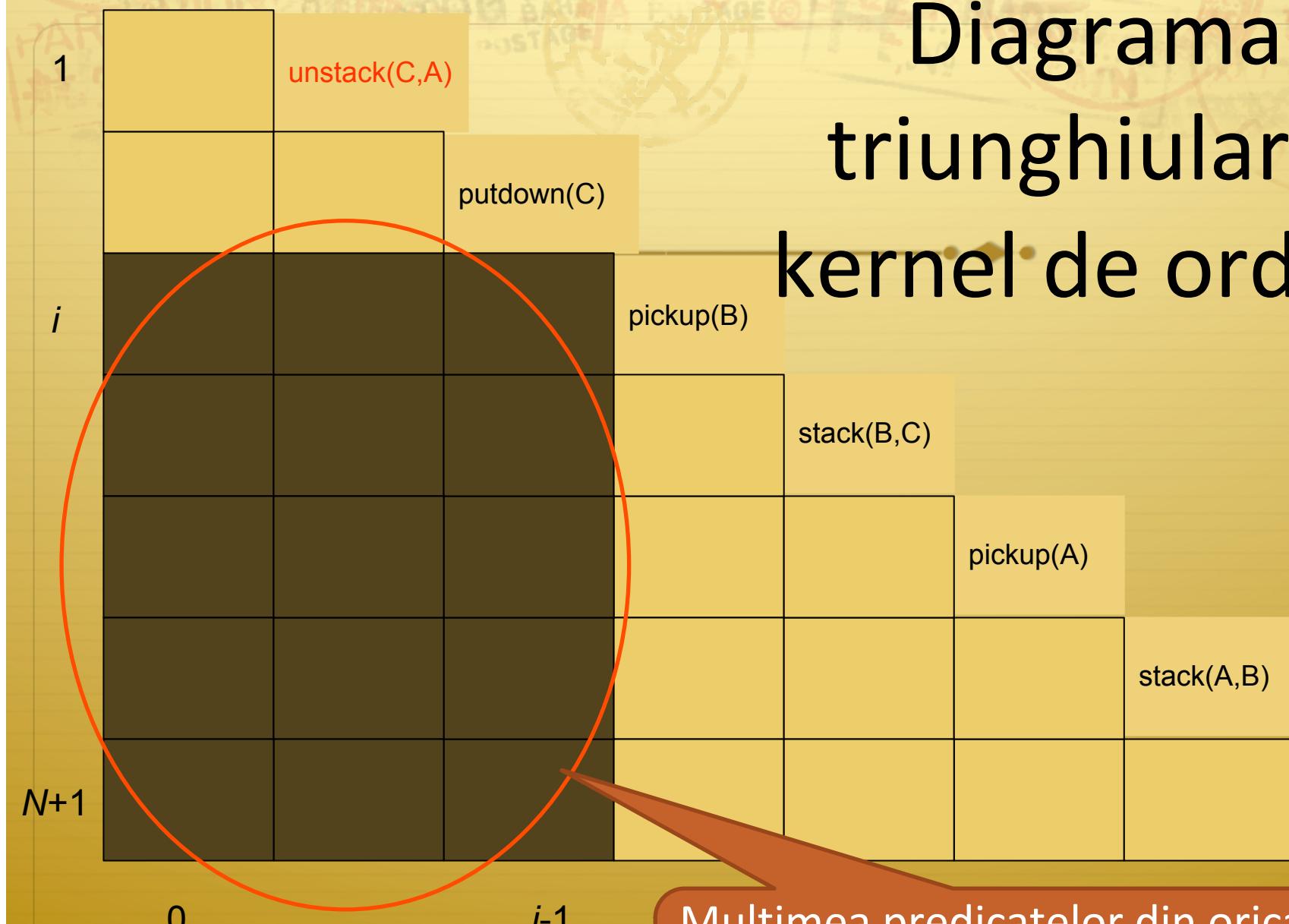
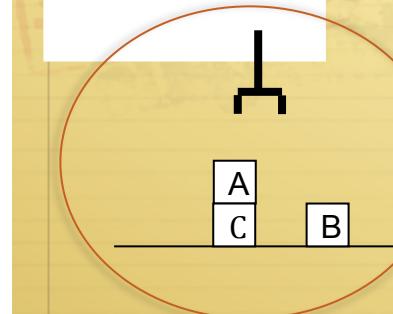


Diagrama triunghiulară: kernel de ordin i

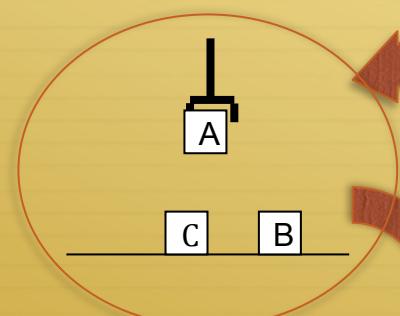


Mulțimea predicatelor din oricare dreptunghi $[i,N+1] \times [0,i-1]$: starea de după execuția regulii $i-1$

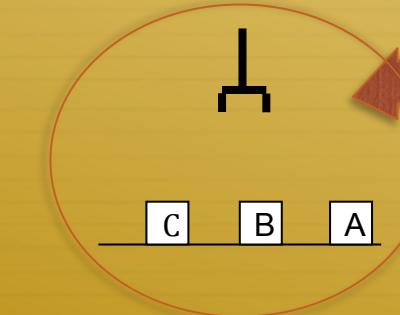
Accident!



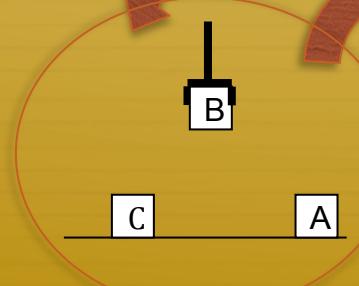
1: unstack(A,C)



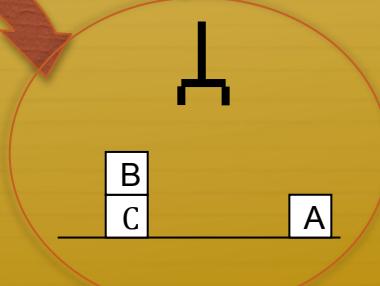
2: putdown(A)



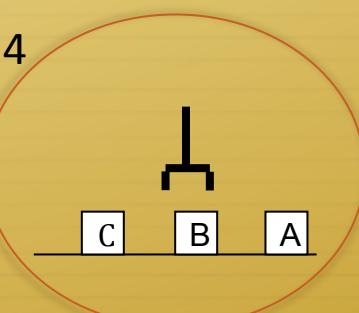
3: pickup(B)



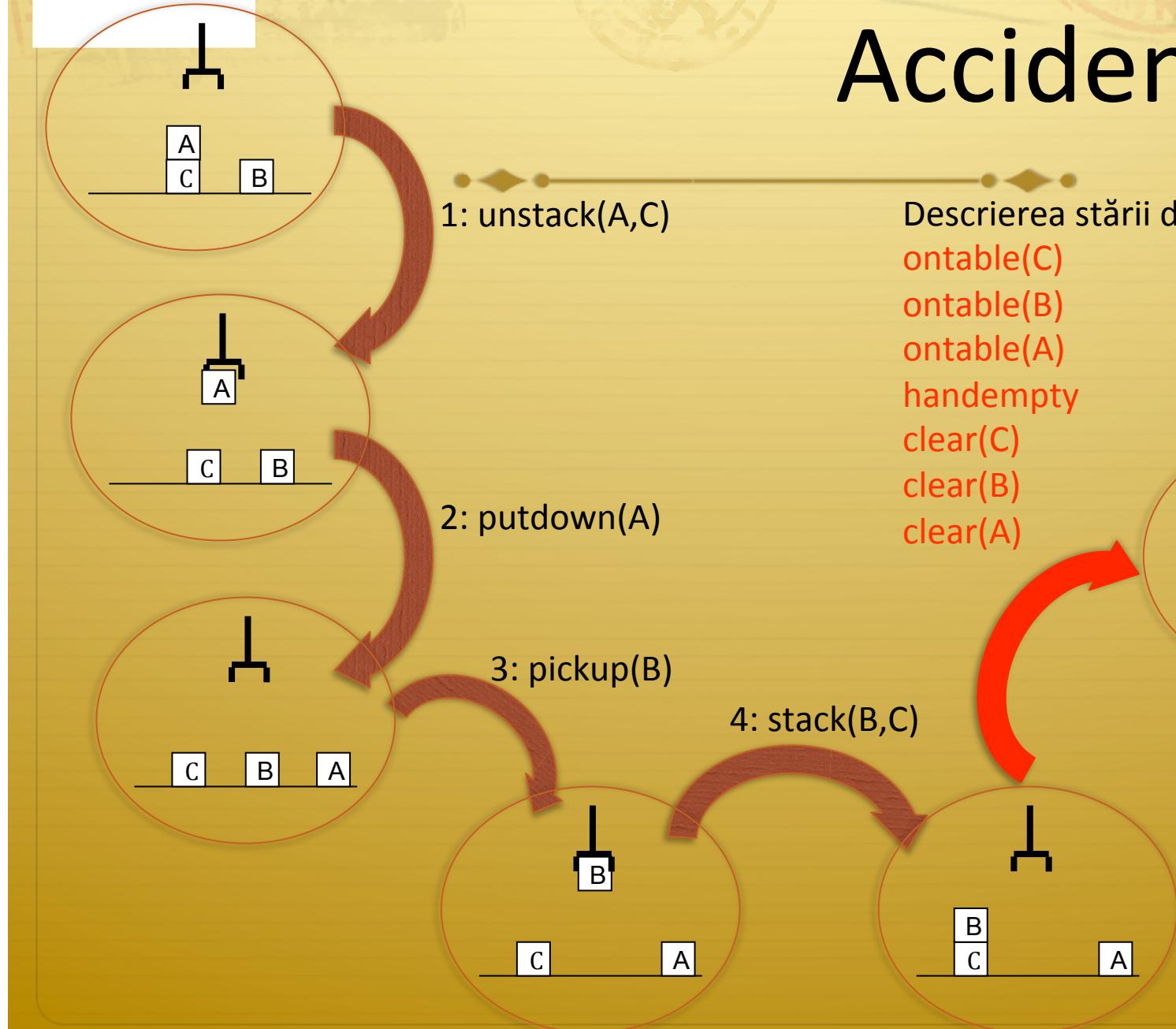
4: stack(B,C)



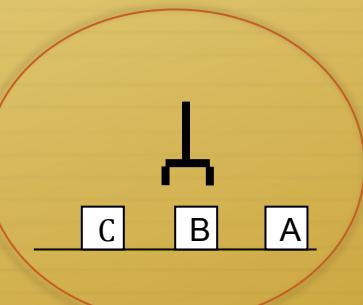
ACCIDENT: după pasul 4
mâna dărâmă stiva!



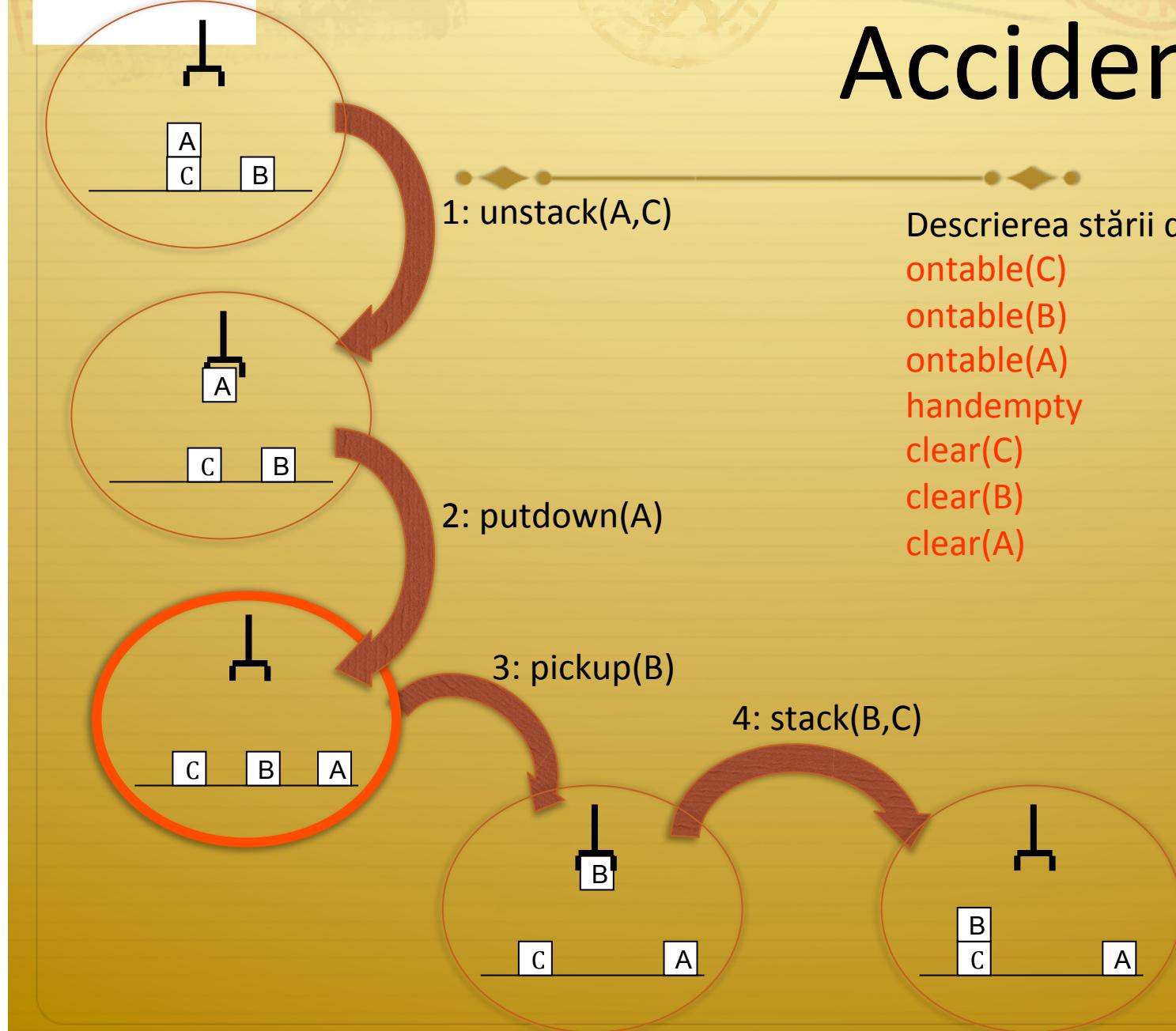
Accident!



Descrierea stării de după accident:
ontable(C)
ontable(B)
ontable(A)
handempty
clear(C)
clear(B)
clear(A)



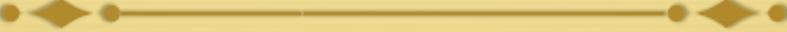
Accident!



Descrierea stării de după accident:

ontable(C)
ontable(B)
ontable(A)
handempty
clear(C)
clear(B)
clear(A)

Concluzia: care e conduită după un accident?

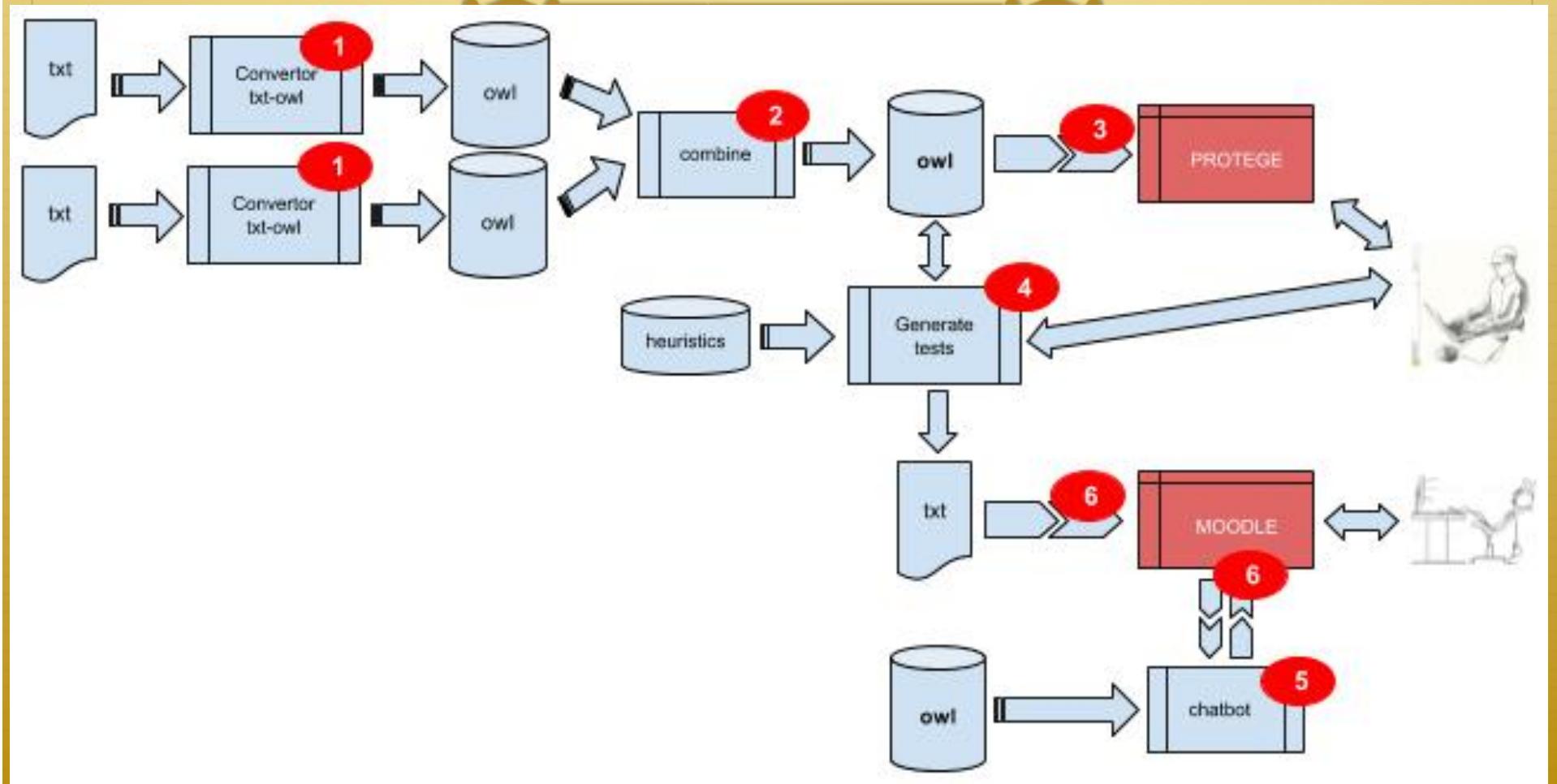


- ❖ Starea de după accident:
 - ❖ se regăsește în mulțimea de kerneluri => reia execuția de acolo!
 - ❖ nu se regăsește => construiește un nou plan cu această stare ca stare initială! => execută planul!



Subiectul 11:
Arhitectura unui sistem de generare de teste, euristici și măsuri de aliniere a termenilor în fuziunea ontologilor.

Proiectul: o arhitectură



Fuziunea de ontologii – definiție

- ❖ Procesul de fuziune ontologică primește în intrare două (sau mai multe) ontologii sursă și returnează o ontologie care combină ontologiile sursă date.

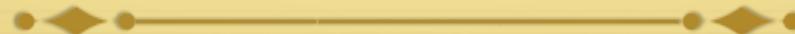
Gerd Stumme, Alexander Maedche. Ontology Merging for Federated Ontologies on the Semantic Web

Abordări

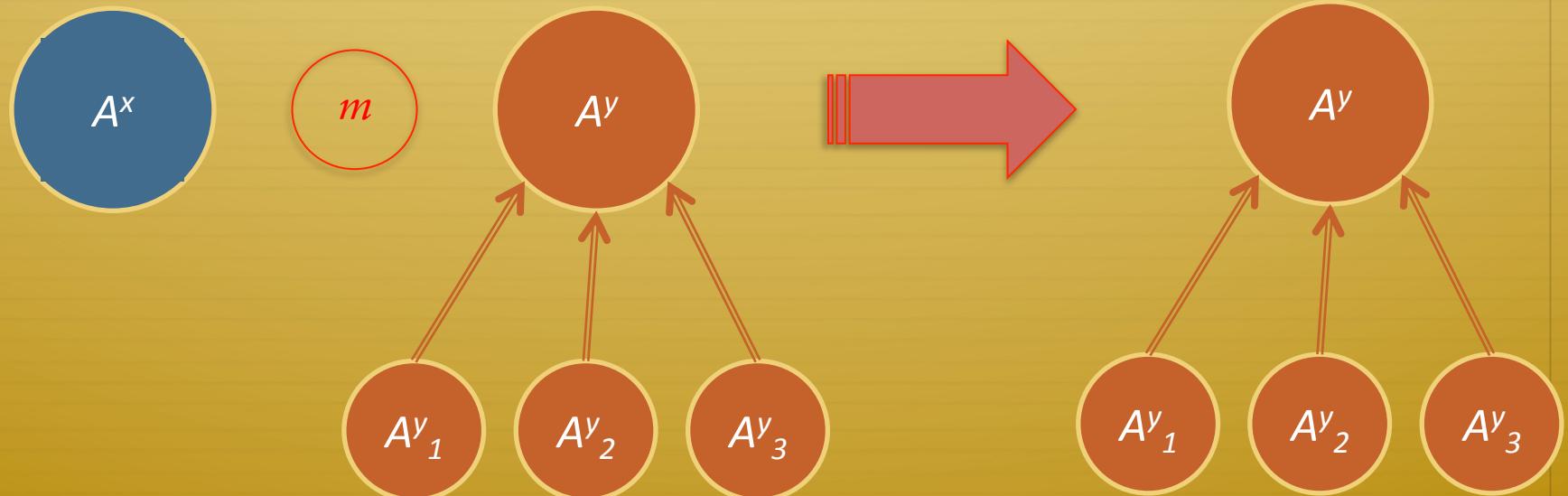


- Abordările se bazează pe euristici de potrivire sintactică și semantică care derivă din comportamentul inginerilor ontologi atunci când se confruntă cu sarcina de a îmbina ontologii, i. e. se simulează comportamentul uman.
- Tehnici statistice, care judecă similaritatea conceptelor și asemănarea brută a instanțelor, prin metrii de șiruri textuale și cunoștințe de natură semantică.

Cum se pot combina termenii?

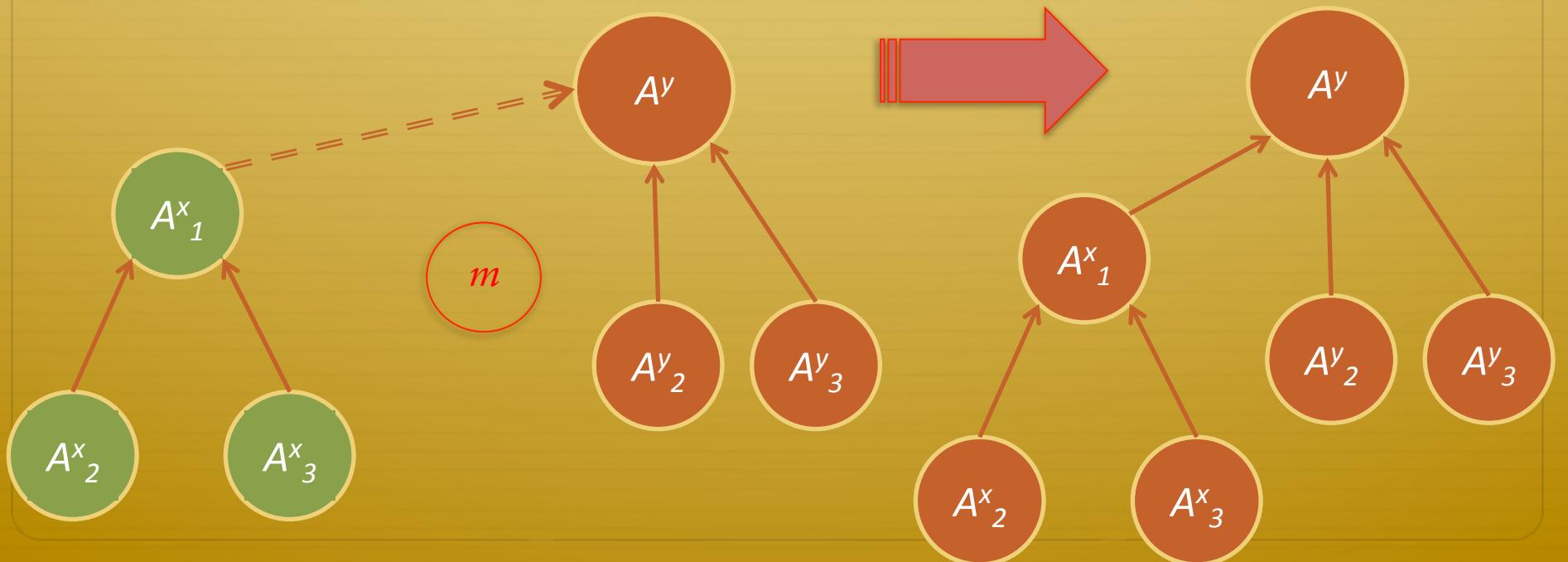


- ❖ Fiind date doi termeni apropiati, cate unul din fiecare ontologie:
 - ❖ Cei doi termeni sunt echivalenți → ei pot fi direct aliniați;

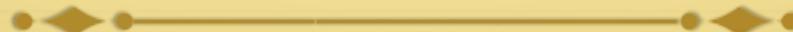


Cum se pot combina termenii?

- ❖ Fiind dați doi termeni, câte unul din fiecare ontologie:
 - ❖ Un termen este mai general decât celălalt → termenul mai specific (și toți subordonații, posibil și frații lui) pot fi integrați sub termenul mai general;



Cum se pot combina termenii?



- Fiind dați doi termeni, câte unul din fiecare ontologie:
 - Termenii sunt incompatibili (i.e., identificarea acestora ar cauza probleme de definire și relaționale între ceilalți termeni) - caz în care:
 - (1) unul dintre termeni trebuie respins și nu trebuie încorporat,
 - (2) unul dintre termeni și alții care depind de el trebuie să fie redefiniți,
 - (3) trebuie creată o "microteorie" separată, în care termenii și toți ceilalți termeni care depind de ea există în paralel
 - (4) poate fi încorporată o versiune mai slabă a termenului infracțional, fără definițiile sau relațiile care au cauzat inconsecvența

Sugestii de euristici de aliniere

1. Potriviri pe șiruri de litere, e.g.:

- ★ Potriviri ale numelor de concepte (*cognate matching*): nume suficient de asemănătoare (în aceeași limbă) sunt dovezi că dezvoltatorii consideră conceptele similare.
- ★ Potriviri în definiții (prin procesare de text și măsuri de suprapuneri): definiții similare în limbaj natural ar trebui fie considerate, de asemenea, dovezi ale similarității conceptelor.

Sugestii de euristici de aliniere

1. Potriviri pe şiruri de litere, e.g.:

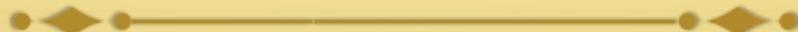
- ❖ Potriviri ale numelor de concepte (*cognate matching*): nume suficient de asemănătoare (în aceeași limbă) sunt dovezi că dezvoltatorii consideră conceptele similare.
- ❖ Potriviri în definiții (prin procesare de text și măsuri de suprapuneri): definiții similare în limbaj natural ar trebui fie considerate, de asemenea, dovezi ale similarității conceptelor.

Sugestii de euristici de aliniere

2. Potrivirile ierarhice exploatează structura de taxonomizare a ontologiilor. Ele includ:

- ❖ Filtrarea ambiguității prin superconcepte partajate: atunci când un concept poate fi aliniat la mai multe alternative, se iau în considerare cele ale căror superconcepte sunt cumva aliniate la superconcepțele conceptului țintă.
- ❖ Măsuri bazate pe distanțe semantice (număr de legături) (v. (Agirre et al., 1994)).

Măsuri de aliniere



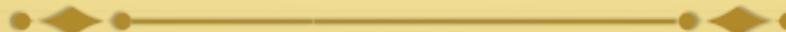
- Potrivire de nume (*cognate match*): compară numele N1 și N2 ale două concepte.
 - Consideră subșiruri descrescătoare ale lui N1, tăind din stânga. Numele formate din cuvinte compuse sunt împărțite în cuvinte separate, se întoarce scorul maxim. Numele mai mici de 3 litere sunt ignorate.
 - NAMESCORE: = numărul de litere potrivite la pătrat + 20 de puncte dacă cuvintele sunt exact egale sau 10 puncte dacă cuvintele coincid la sfârșit

Măsuri de aliniere



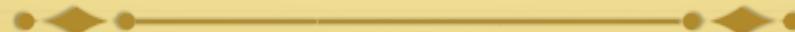
- ❖ Potriviri pe definiții: compară definițiile în engleză D1 și D2 ale două concepte. Mai întâi, ambele definiții sunt separate în cuvinte separate (se îndepărtează apostroafele, limioarele de unire etc.) și toate cuvintele sunt lematizate.
 - ❖ definiția lui M@FOOD: ("any" "substance" "that" "can" "be" "metabolized" "organism" "give" "energy" "build" "tissue")
 - ❖ apoi, se calculează 3 valori:
 - ❖ strength = raportul dintre numărul de cuvinte care apar în ambele definiții și numărul de cuvinte ale definiției cele mai scurte,
 - ❖ reliability = numărul de cuvinte comune,
 - ❖ defscore = strength * reliability: .
 - ❖ DEFSCORE := $(\text{Shared}(D1, D2) / \min\{D1, D2\}) * \text{Shared}(D1, D2)$

Măsuri de aliniere



- Potrivire TAXONOMICĂ (între ontologiile SENSUS și MIKROKOSMOS): pentru un anumit concept SENSUS, colectează toate conceptele din MIKROKOSMOS care sunt "mai apropiate" de 10 link-uri de el. Algoritmul traversează taxonomia atât în direcțiile superconcept cât și în subconcepțe.
 - Scorul de potrivire este dat de inversa link-distanței:
 - **TAXSCORE** := 1 / number-of-links

Combinarea scorurilor



- Caracteristicile formulelor de combinare:
 - să crească cu valori în creștere ale NAME, DEF și TAX
 - să normalizeze scorurile euristicilor
 - să diminueze tendința scorurilor NAME de a crește rapid
 - să atenuarea tendința scorurilor de TAXONOMIE de diminuare rapidă
 - să întoarcă un scor nenul dacă cel puțin o heuristică întoarce un scor nenul
- $\text{SCORE} := \text{sqrt}(\text{NAMESCORE}) * \text{DEFSCORE} * (10 * \text{TAXSCORE})$

cu grijă că dacă NAMESCORE sau DEFSCORE sunt zero, ele sunt înlocuite prin 1, și dacă TAXSCORE e 0, el e înlocuit prin 0.01.

Uzual, scorurile de aliniere se plasează în scara 0 – 16.



• ◆ • ————— • ◆ •

Subiectul 12:

Patternuri lexicale pentru determinarea relației de hiponimie/hipernimie, patternuri de generare de teste.

Patternuri lexicale



$NP_0 \dots \text{ such as } \{NP_1, NP_2 \dots \text{ (and | or)}\} NP_n$

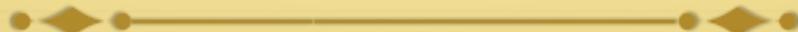
implică

for all $NP_i, 1 < i < n$, hyponym(NP_i, NP_0)

Din exemplul de mai sus rezultă:

hyponym ("Barmbara ndang", "bow lute")

Patternuri lexicale



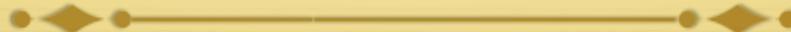
such NP as {NP ,}* {(or | and)} NP

*... works by such authors as Herrick,
Goldsmith, and Shakespeare.*

=>

```
hyponym("author", "Herrick")
hyponym("author", "Goldsmith")
hyponym("author", "Shakespeare")
```

Patternuri lexicale



NP {, NP} * {,} or other NP

Bruises, wounds, broken bones or other injuries . . .

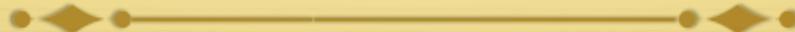
=>

hyponym("bruise", "injury")

hyponym ("wound", "injury ")

hyponym("broken bone", "injury")

Patternuri lexicale



NP {, NP}* {,} and other NP

*... temples, treasuries, and other
important civic buildings.*

=>

hyponym("temple", "civic building")
hyponym("treasury ", "civic building")

Patternuri lexicale



NP {,} including {NP {,}} * {or | and} NP

All common-law countries, including

Canada and England ...

=>

hyponym("Canada", "common-law country")

hyponym ("England", "common-law country")

Patternuri lexicale



NP {,} especially {NP ,}* {or] and} NP

. . . most European countries, especially
France, England, and Spain.

=>

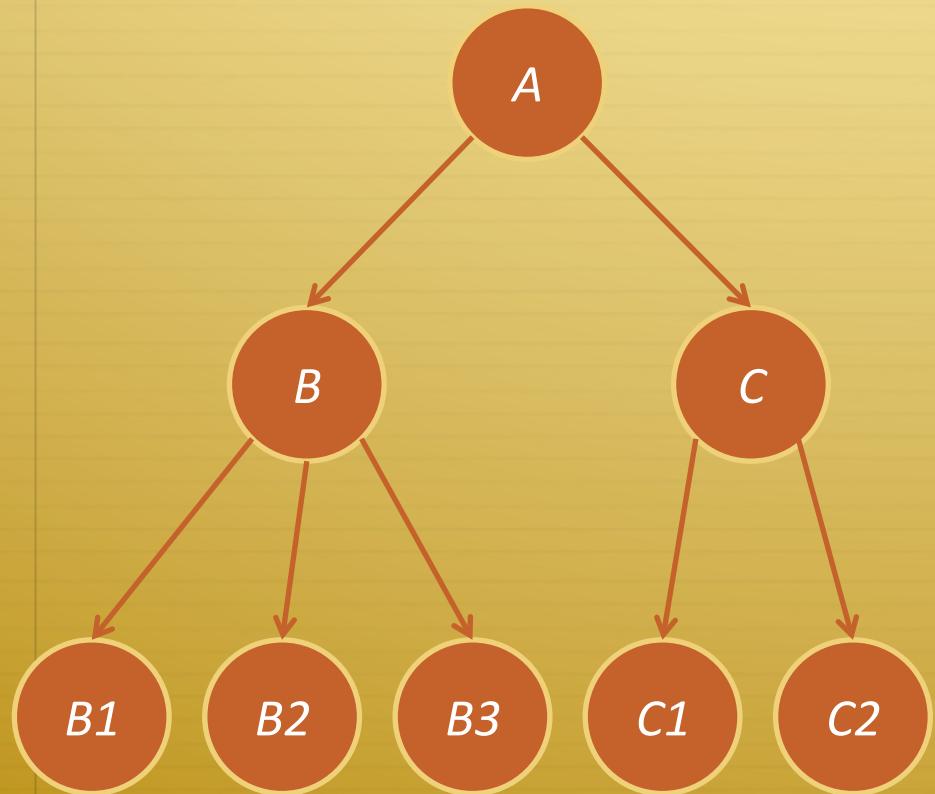
hyponym("France", "European country")
hyponym("England", "European country")
hyponym("Spain", "European country")

Patternuri în generarea testelor

Žitko B, Stankov S, Rosić M, Grubišić A. (2009) Dynamic test generation over ontology-based knowledge representation in authoring shell. *Expert Systems with Applications*. 36:8185–8196.

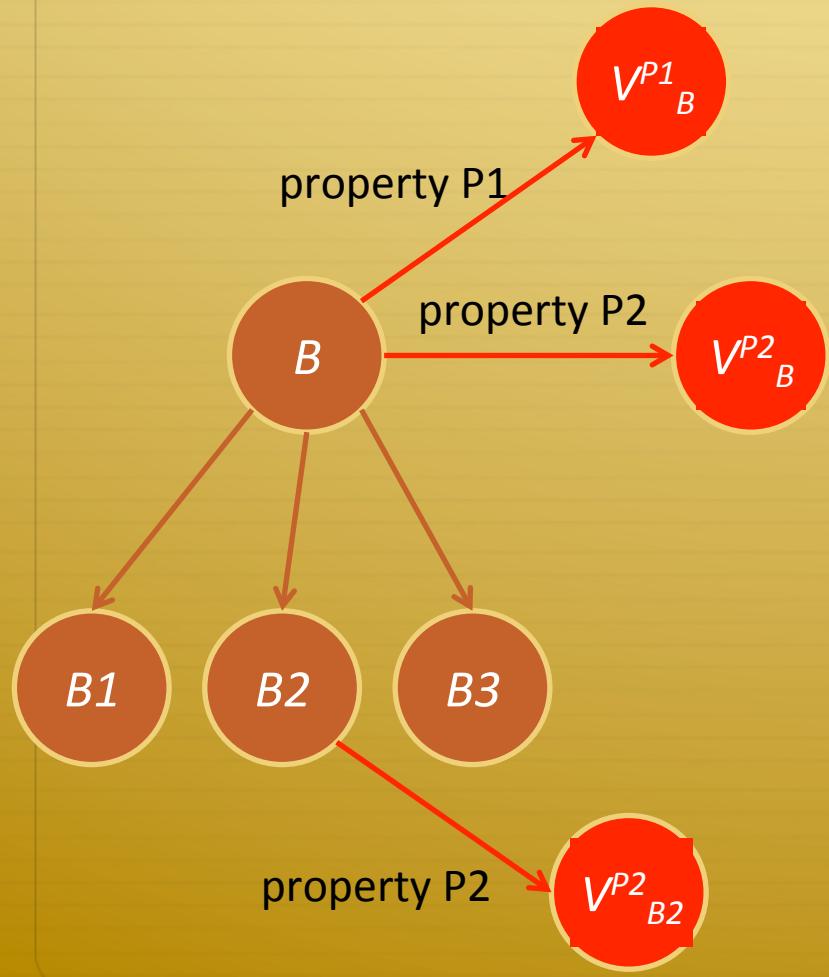
- ❖ Teste generate pe baza cunoștințelor de domeniu exprimate în ontologii OWL:
 - ❖ un număr de şabloane definite pentru întrebări sunt utilizate de sistem pentru a genera elementele de testare

Exemple de patternuri care generează teste



Care dintre următoarele elemente:
shuffle(someOf(desc(B))),
someOf(desc(siblingOf(B)))
sunt B-uri?

Exemple de patternuri care generează teste



Care dintre următoarele proprietăți:
setOfPropertiesOf(B) caracterizează
random(descOf(B)) și care este
valoare ei?