



Inteligentă Artificială

Universitatea Politehnica Bucuresti
Anul universitar 2020-2021

Adina Magda Florea



Curs nr. 6

Sisteme bazate pe reguli

- Reprezentarea prin reguli
- Structura SBR
- Ciclul de inferenta al unui sistem bazat pe reguli
- Strategia de control
- SBR cu inlantuire inainte
- SBR cu inlantuire inapoi
- Modelul cunoștințelor incerte din MYCIN – factori de certitudine

1. Reprezentarea prin reguli

- Reprezentare modulara a cunostintelor procedurale
- Cunostinte procedurale in maniera declarativa
- Permit atasare de actiuni (functii, proceduri)
- Se pot combina cu reprezentari structurate ale cunostintelor, de ex. ontologii
- Multiple limbaje bazate pe reguli

Limbaje bazate pe reguli

Precursori

- OPS5 - Official Production System
- MYCIN

Limbaje bazate pe reguli

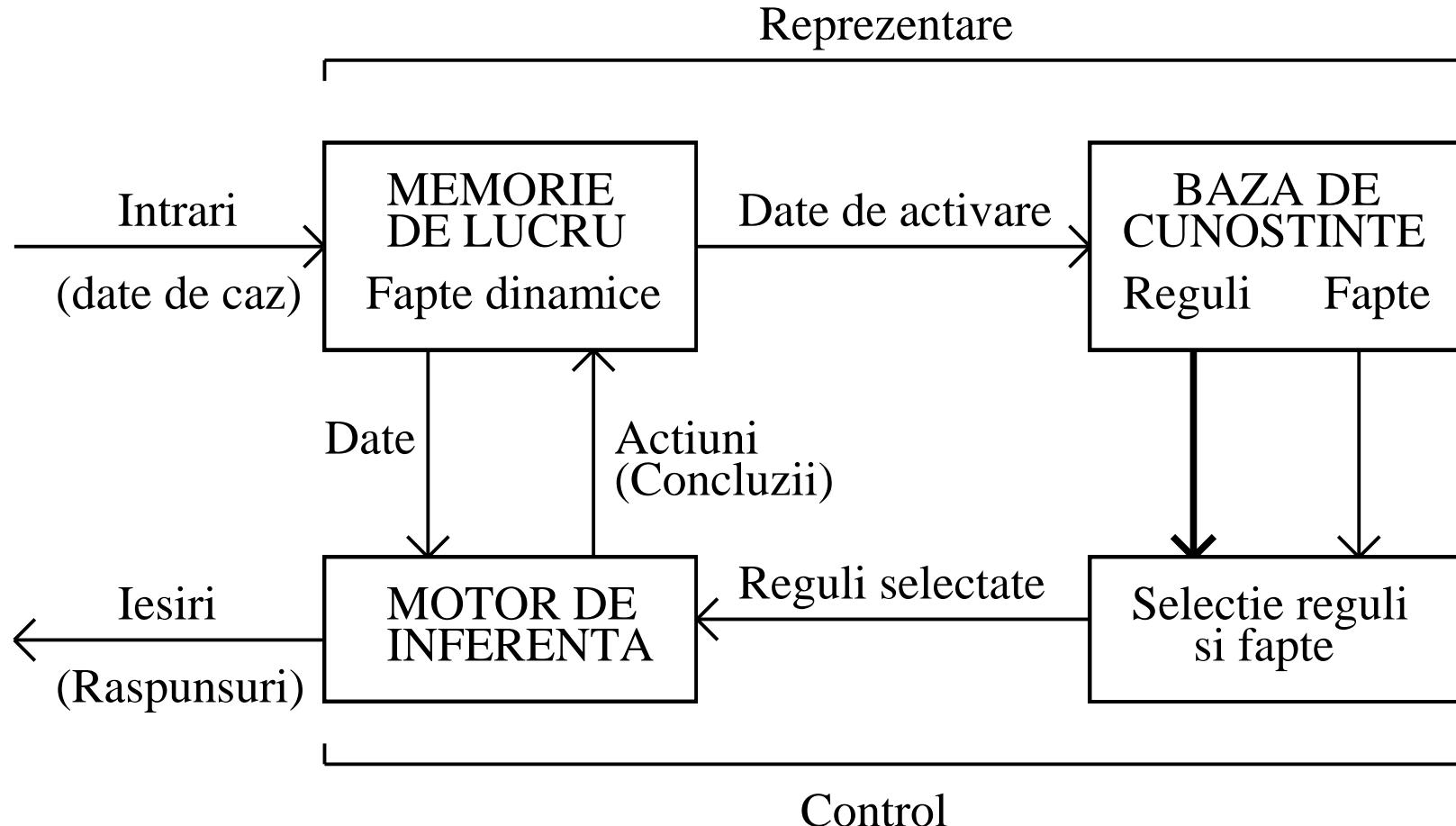
Noi, moderne

- CLIPS - **C Language Integrated Production System**
- Jess – **Java Expert System Shell** - written in Java, superset of CLIPS
- RuleML – **Rule Markup Language** (Rule Markup Initiative)
- SWRL - **Semantic Web Rule Language**
- Drools - **Business Rules Management System**
<https://www.drools.org/>
- Legal XML <http://www.legalxml.org/>
- Declarative AI 2020 - Rules, Reasoning, Decisions, and Explanations <https://2020.declarativeai.net/>

Forme de reguli

- **Reguli de inferenta**
 - **if** conditie(i) **then** concluzie/actiune
- Utilizare
 - obtinerea de noi cunostinte
 - executarea actiunilor in functie de conditii
- **Reguli reactive**
 - **on** eveniment **if** conditie(i) **then** (executa) actiune
- Utilizare
 - triggere in SMBD
 - detectarea exceptiilor
 - reguli de integritate
- Se pot combina

2. Structura SBR



3. Ciclul de inferenta al unui sistem bazat pe reguli

- Identificare
- Selectie
- Executie

Ciclul de inferenta al unui sistem bazat pe reguli - cont

Algoritm: Functionarea unui sistem bazat pe reguli

1. $ML \leftarrow$ Date de caz

2. **repeta**

2.1. Executa identificare intre ML si BC si construieste multimea de conflicte a regulilor aplicabile (MC)

2.2. Selecteaza o regula dupa un criteriu de selectie

2.3. Aplica regula prin executia partii drepte a regulii

pana nu mai sunt reguli aplicabile **sau**

memoria de lucru satisface conditia de stare scop **sau**

o cantitate predefinita de efort a fost epuizata

sfarsit.

4. Strategia de control

- Criteriile de selectie din MC
- Directia de aplicare a regulilor

4.1 Criteriile de selectie din MC

- Selectia primei reguli aplicabile
- Alegerea unei reguli din multimea de conflicte
 - Preferinte bazate pe natura regulilor
 - Specificitate*
 - Momentului folosirii anterioare*
 - Preferinte bazate pe obiectele identificate
 - Preferinte bazate pe natura starilor

Criteriile de selectie din MC - cont

■ Utilizarea metaregulilor

daca o regula are conditiile A si B **si**

regula refera {nu refera} X

{ de loc/

numai in partea stanga/

numai in partea dreapta }

atunci regula va fi in special utila

{ probabil utila/

probabil inutila/

sigur inutila }

■ Aplicarea tuturor regulilor din multimea de conflicte

4.2 Directia de aplicare a regulilor

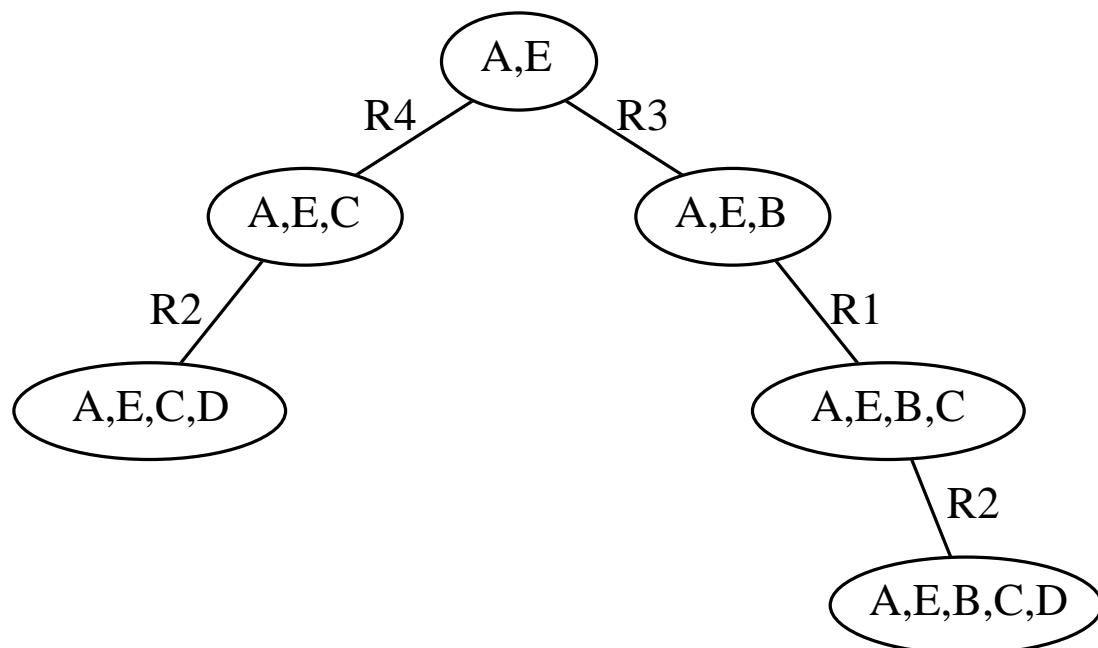
Continut initial al memoriei de lucru: A,E
Stare scop: D

$$A,E \xrightarrow{R3} A,E,B \xrightarrow{R1} A,E,B,C \xrightarrow{R2} A,E,B,C,D$$

(a) Inlantuire inainte

- R1: daca A&B atunci C
- R2: daca C atunci D
- R3: daca A atunci B
- R4: daca A&E atunci C

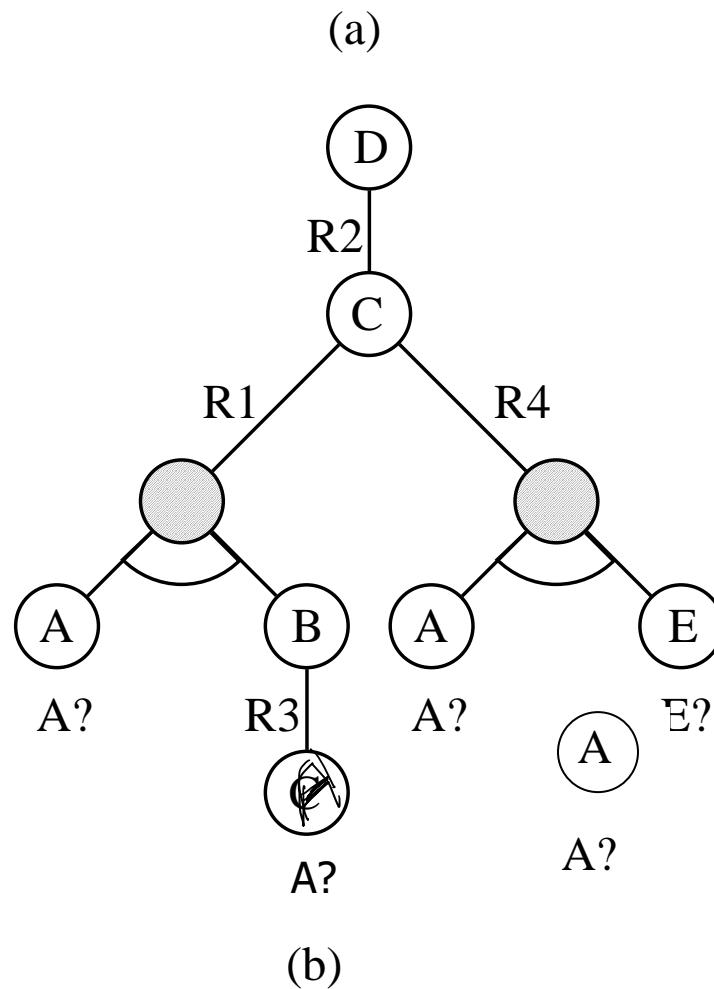
(a)



Continut initial al memoriei de lucru: A,E
Stare scop: D

(b) Înlăturare înapoi

$$D? \xrightarrow{R2} C? \xrightarrow{R1} A?, B? \xrightarrow{R3} A?, E?$$



- R1: daca A&B atunci C
- R2: daca C atunci D
- R3: daca A atunci B
- R4: daca A&E atunci C

Algoritm: Determinarea unei valori cu inlantuire inainte a regulilor (a)

DetValoareD(A)

1. **daca** A are valoare
atunci intoarce SUCCES
2. Construieste multimea de conflicte, MC, si initializeaza Efort
3. **daca** $MC = \{ \}$
atunci intoarce INSUCCES
4. Alege o regula $R \in MC$ dupa un criteriu de selectie
5. Aplica regula R
 - Adauga faptul din concluzia regulei R la ML (daca nu exista deja)
 - sau executa actiunea din concluzie

6. **daca** A are valoare ($A \in ML$)
atunci intoarce SUCCES
 7. $MC \leftarrow MC - R$
 8. $MC \leftarrow MC \cup$ Noile reguli aplicabile
 9. Actualizeaza Efort
 10. Daca $Efort = \{ \}$ atunci **intoarce** INSUCCES
 11. **repeta** de la 3
- sfarsit.**

Forward
chaining

Algoritm: Determinarea unei valori cu inlantuirea inapoi a regulilor (b)

DetValoare(A)

1. Construieste multimea regulilor care refera A in concluzie, MC
2. **daca** MC = { }

atunci

- 2.1. Intreaba utilizatorul valoarea lui A
- 2.2. **daca** se cunoaste valoarea lui A
atunci depune in ML aceasta valoare
altfel intoarce INSUCCES

*Backward
Chaining*

3. **altfel**

- 3.1. Alege o regula dupa un criteriu de selectie
- 3.2. **pentru** fiecare ipoteza Ij, j=1,N, a premisei lui R **executa**
 - 3.2.1. Fie Aj atributul referit de ipoteza Ij
 - 3.2.2. **daca** Aj are valoare
atunci atunci evalueaza adevarul ipotezei Ij
 - 3.2.3. **altfel**
 - i. **daca** DetValoare(Aj) = SUCCES
atunci evalueaza adevarul ipotezei Ij
 - ii. **altfel** considera ipoteza Ij nesatisfacuta

- 3.3. **daca** toate ipotezele I_j , $j=1, N$ sunt satisfacute
atunci depune in ML valoarea lui A dedusa pe baza concluziei R
4. **daca** A are valoare (in ML)
atunci intoarce SUCCES
5. **altfel**
- 5.1 $MC \leftarrow MC - R$
- 5.2 **repeta** de la 2
- sfarsit.**
-

- 1'. **daca** A este data primara
atunci
- 1'.1. Intreaba utilizatorul valoarea lui A
- 1'.2. **daca** se cunoaste valoarea lui A
 atunci - depune in ML valoarea lui A
 - **intoarce** SUCCES
- 1'.3. **altfel intoarce** INSUCCES

5. SBR cu înlantuire inainte

Familia OPS5

- OPS5
- ART
- CLIPS
- Jess
- Familia Web Rule languages
 - În special RuleML și SWRL (OWL + Rule language)
 - Interoperabilitatea regulilor
- Drools

SBR cu înlantuire înainte

- Ciclul recunoastere-actiune:
 - Match
 - Select
 - Act
- WME = working memory element
 - Identificat printr-un "time tag"
- Instantiere: multime de WME care satisfac o regula
- Multime de conflicte (MC)
- Rezolvarea conflictelor

SBR cu inlantuire inainte

- Verifica la fiecare ciclu care reguli sunt aplicabile
- Match – cam 80% din timpul ciclului recunoastere-actiune
- WME
- Fiecare WME este comparat cu fiecare conditie din fiecare regula

SBR cu inlantuire inainte - eficienta

Cresterea eficientei implementarii

Partitionarea regulilor in functie de o conditie

- Elimin necesitatea sa verific la fiecare ciclu care reguli sunt aplicabile

Forward chaining incremental

- Algoritmul RETE

SBR cu inlantuire inainte - RETE

- RETE match algorithm (Forgy)
 - Salveaza starea de match intre 2 cicluri recunoastere-actiune
 - La crearea unui WME, acesta este comparat cu toate elementele conditie din program si este memorat impreuna cu fiecare element conditie cu care a identificat =>
 - Numai schimbarile incrementale din ML sunt identificate in fiecare ciclu

Exemplu JESS

```
(deftemplate student
  (slot name)
  (slot sex)
  (slot placed_in)
  (slot special_considerations (default no))
```

```
(deftemplate room
  (slot number)
  (slot capacity (type INTEGER)(default 4))
  (slot sexes_are)
  (slot vacancies (type INTEGER))
  (multislot occupants))
```

Exemplu JESS

```
(assert (room (number 112) (capacity 4) (vacancies 4)
              (sexes_are nil) (occupants nil)))
(assert (student (name Mihai) (placed_in nil) (sex M)))
```

WMEs

| | |
|------------------------|------------|
| (room 112 4 4 nil nil) | time tag1 |
| (student Mihai nil M) | time tag 2 |

Stud/cam in Jess

```
(defrule assign-student-empty-room
```

```
?unplaced_student <- (student (name ?stud_name)  
                           (placed_in nil)  
                           (sex ?gender))
```

```
?empty_room <- (room (number ?room_no)  
                        (capacity ?room_cap)  
                        (vacancies ?max_size))  
(test (= ?room_cap ?max_size))
```

```
=>
```

```
(modify ?unplaced_student (placed_in ?room_no))  
(modify ?empty_room (occupants ?stud_name) (sexes_are ?gender)  
          (vacancies (-- ?max_size)))
```

```
)
```

Multimea de conflicte

(defrule assign-student-empty-room

(student (name ?stud_name)

(placed_in nil)

(sex ?gender))

(room (number ?room_no)

(capacity ?room_cap)

(vacancies ?max_size))

(test (= ?room_cap ?max_size))

=> ...

Instantieri (MC)

assign-student-empty-room 41 9

assign-student-empty-room 41 17

assign-student-empty-room 52 9

assign-student-empty-room 52 17

WMEs

41 (student name Mary sex F placed_in nil)

52 (student name Peter sex M placed_in nil)

9 (room number 221 capacity 1 vacancies 1)

12 (room number 346 capacity 2 vacancies 1)

17 (room number 761 capacity 1 vacancies 1)

Rezolvarea conflictelor

- Diferite strategii
 - **Refractie** = o aceeasi instantiere nu este executata de mai multe ori (2 instantieri sunt la fel daca au acelasi nume de regula si aceleasi time tags, in aceeasi ordine)
 - **Momentul utilizarii** = Se prefera instantierile care au identificat cu WMEs cu cele mai recente time tags (sau invers)
 - **Specificitate** = Au prioritate instantierile cu LHS specifice = nr de valori testate in LHS
 - teste asupra: nume clasa, predicat cu 1 arg constanta, predicat cu un operator variabila legata

+ Left Hand Side
Right Hand Side

Rezolvarea conflictelor

- **Strategia LEX**
 - Elimina din MC instantierile care au fost deja executate
 - Ordoneaza instantierile pe baza momentului utilizarii
 - Daca mai multe instantieri au aceleasi time tags, utilizeaza specificitate
 - Daca mai multe instantieri au aceeasi specificitate alege arbitrar
- **Strategia MEA** – aceeasi dar utilizeaza primul time tag

Construirea MC

Match

- O regula se poate aplica daca conditiile din antecedent sunt satisfacute de WMEs din ML
- Procesul are 2 aspecte:
 - match intra-element
 - match inter-element

(defrule assign-student-room

 (student (name ?stud_name) (placed_in nil)
 (sex ?gender))

 (room (number ?room_no) (sexes_are ?gender)
 (vacancies ?size))

 (test (> ?size 0))

=> ...

RETE - Compilarea regulilor

- Se compileaza conditiile regulilor intr-o **retea de noduri de test** – retea de flux de date
- Există noduri pt teste **intra-element (Alpha nodes)** si **inter-element (Beta nodes)**
- Intrarea – schimbari in ML
- Iesirea – schimbari in MC

RETE - Compilarea regulilor

- Un pointer la noul WME este trecut prin retea, incepand de la nodul radacina
- Fiecare nod actioneaza ca un switch
- Cand un nod primeste un pointer la un WME, testeaza WME asociat. Daca testul reuseste, nodul se deschide si WME trece mai departe
- Altfel nu se intampla nimic
- Daca un WME va trece prin retea, va fi combinat cu alte WME care trec pentru a forma o instantiere in MC
- Pointerii la WME sunt trimisi prin reteaua RETE ca tokens = pointer + stare (assert sau retract)

(defrule cat-job-size

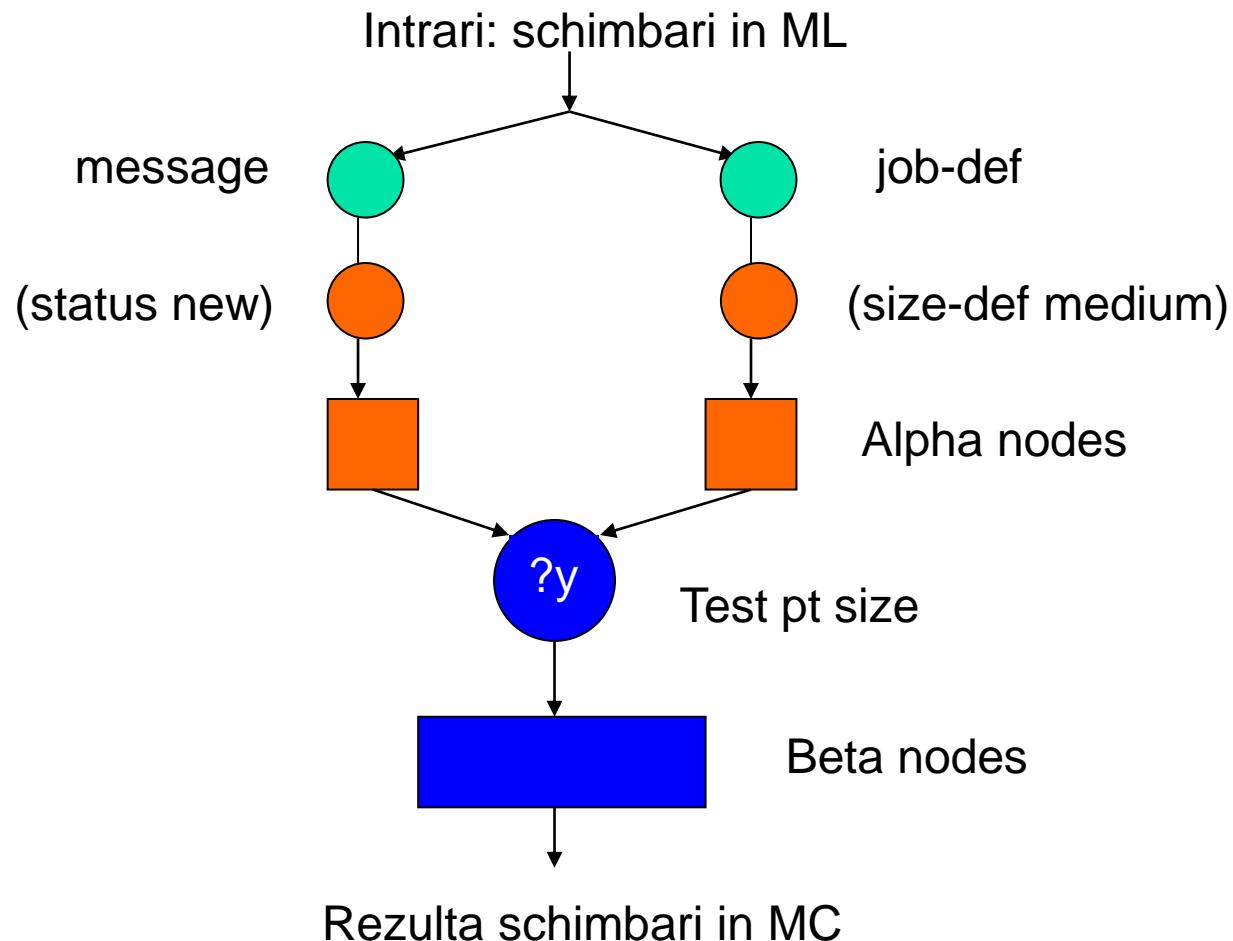
 (message (jobs ?x) (status new) (size ?y))

 (job-def (size-def medium) (size ?y))

=>

 (assert (job (job-name ?x) (job-size ?y))))

Retea RETE pt regula



(defrule show-act

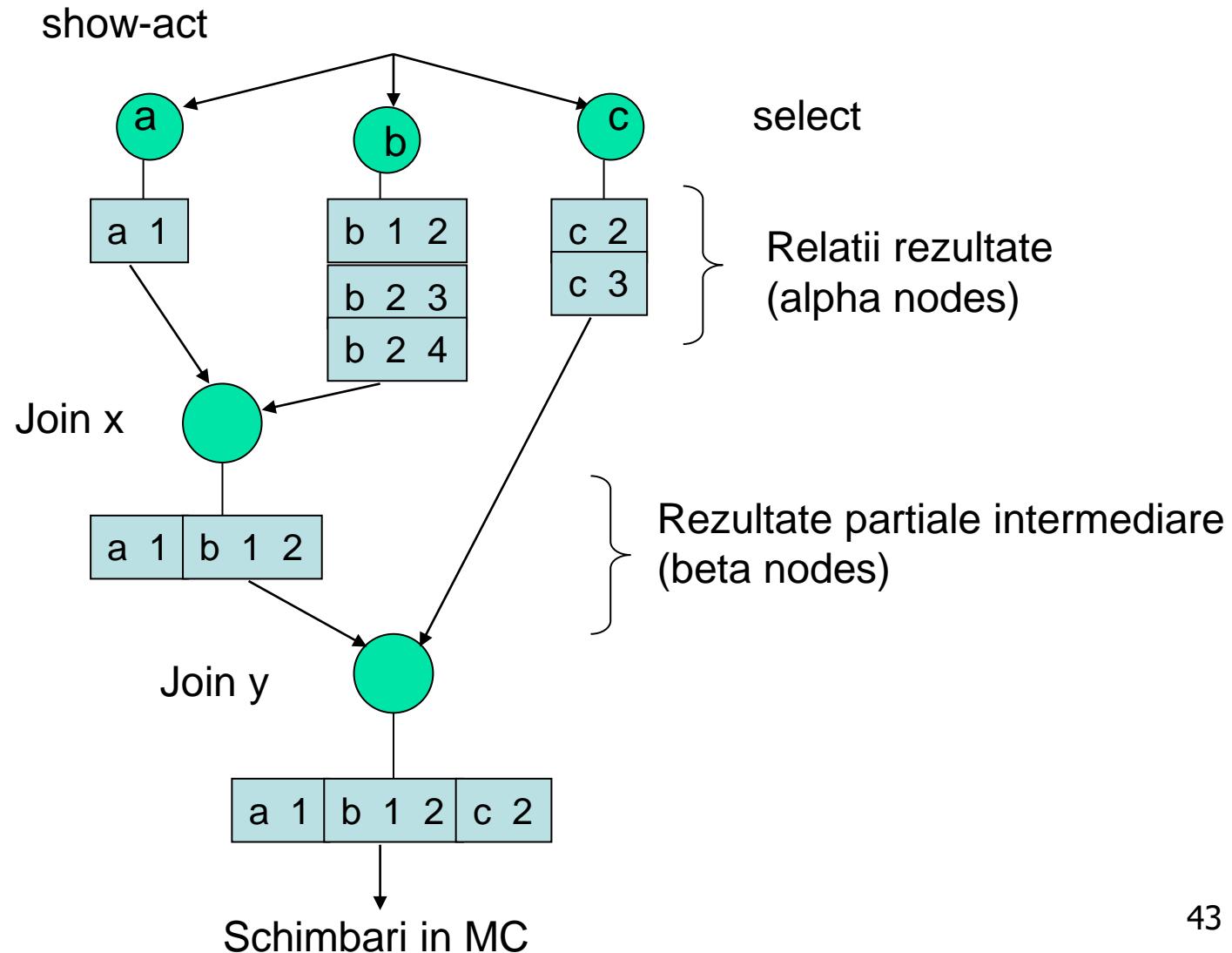
(a ?x)
(b ?x ?y)
(c ?y)

=> ...

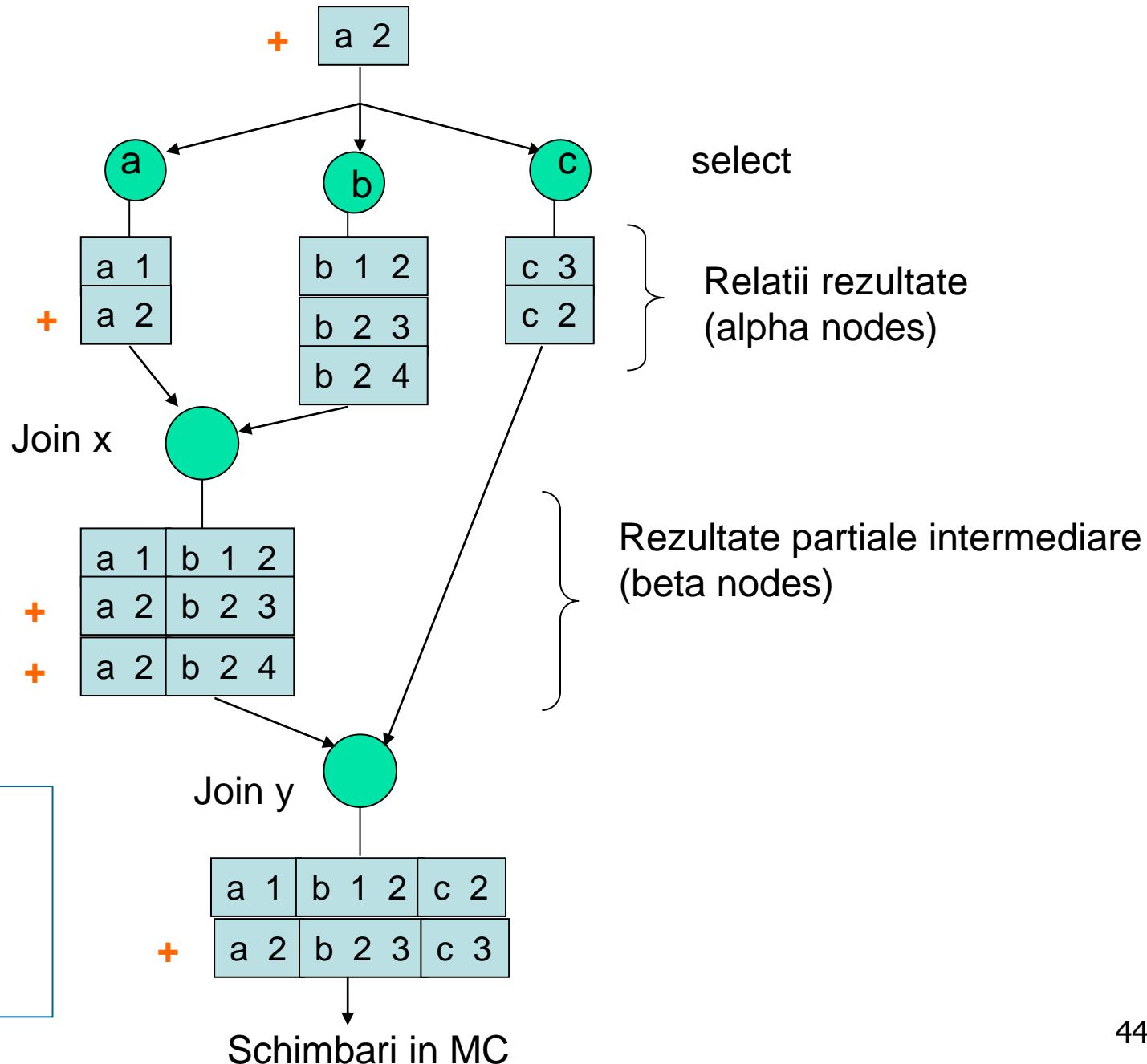
ML

(a 1)
(b 1 2) (b 2 3) (b 2 4)
(c 2) (c 3)

Stare initială a retelei

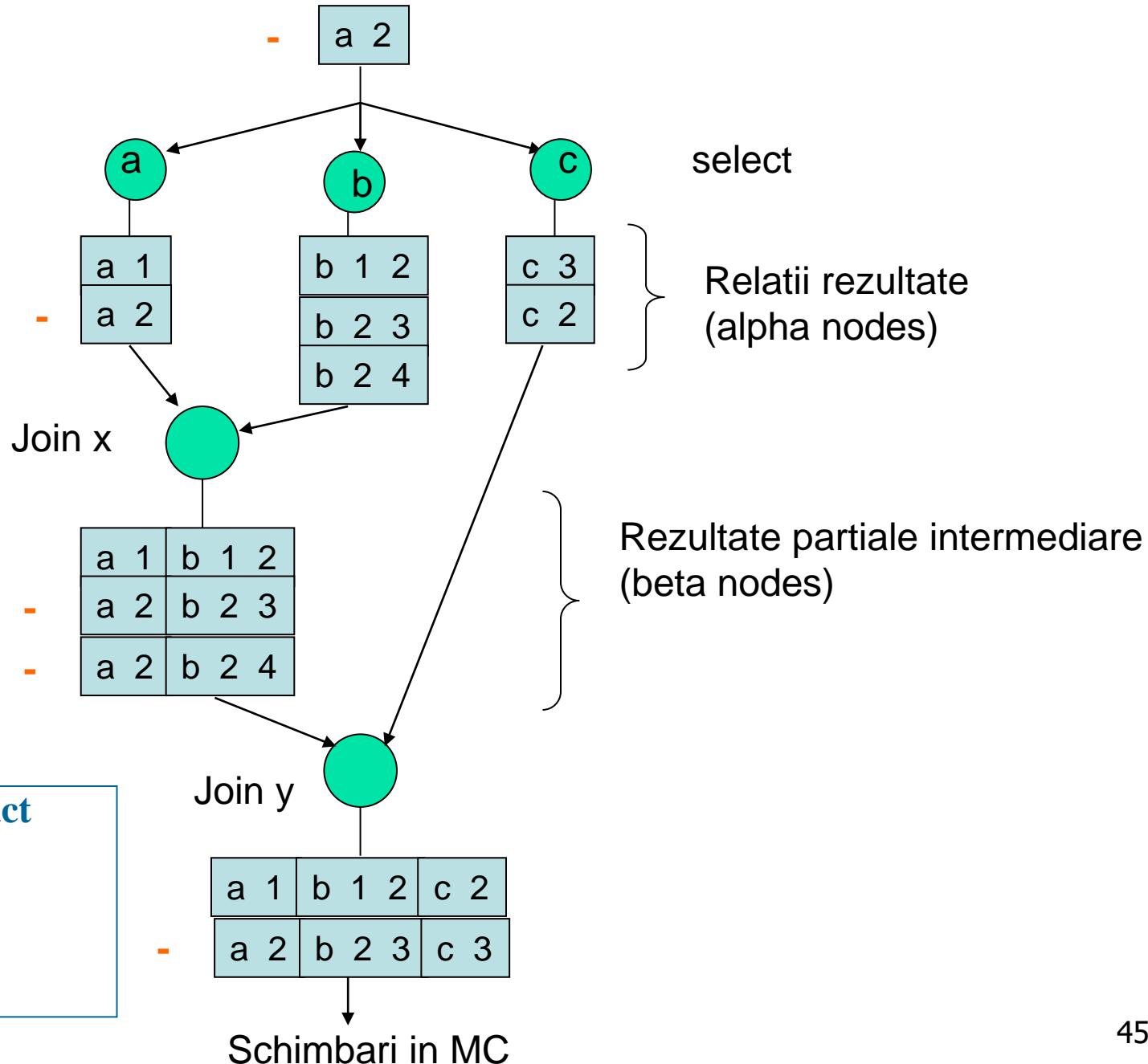


Adaugare WME (assert)



```
(defrule show-act
  (a ?x)
  (b ?x ?y)
  (c ?y)
=> ...)
```

Eliminare WME (retract)

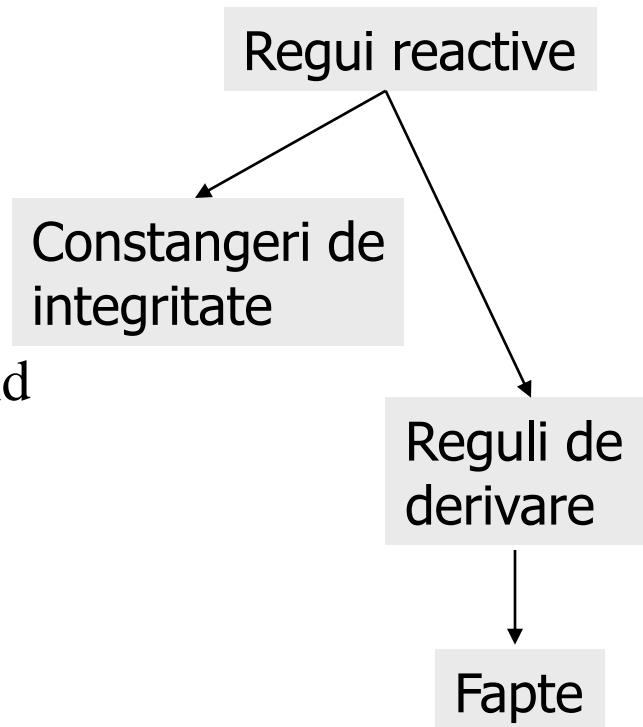


RuleML – pt Web

- RuleML Initiative – ruleml.org (2013)
- RuleML Initiative - dezvolta limbaje bazate pe reguli deschise XML/RDF
- Limbajul RuleML – sintaxa XML pentru reprezentarea cunostintelor sub forma de reguli
- Aceasta permite schimbul de reguli intre diferite sisteme, inclusiv componente software distribuite pe Web si sisteme client-server eterogene
- Integrarea cu ontologii: sistemul de reguli poate sa deriveze/utilizeze cunostinte din ontologie

RuleML

- **Reguli reactive** = Observa/verifica anumite evenimente/conditii si executa o actiune – numai forward
- **Constrangeri de integritate** = reguli speciale care semnaleaza inconsistente cand se indeplinesc anumite conditii – numai forward
- **Reguli de inferenta** (derivare) = reguli reactive speciale cu actiuni care adauga o concluzie daca conditiile (premisele sunt adevarate) - Se pot aplica atat forward cat si backward
- **Fapte** = reguli de inferenta particulare



RuleML

■ Reguli reactive

```
<rule>  <_body><and> prem1 ... premN </and></_body>
        <_head> action </_head>
</rule>
```

■ Constanțe de integritate

```
<ic>    <_body><and> prem1 ... premN </and></_body>
            <_head> inconsistency </_head>
</ic>
```

implementate ca

```
<rule>  <_body><and> prem1 ... premN </and></_body>
        <_head><signal> inconsistency </signal> </_head>
</rule>
```

RuleML

■ Reguli de inferenta/derivare

```
<imp> <_head> conc </_head>
      <_body> <and> prem1 ... premN </and> </_body>
</imp>
```

implementate prin

```
<rule>  <_head> <assert> conc </assert> </_head>
        <_body> <and> prem1 ... premN </and> </_body>
</rule>
```

■ Fapte

```
<atom> <_head> conc </_head> </atom>
```

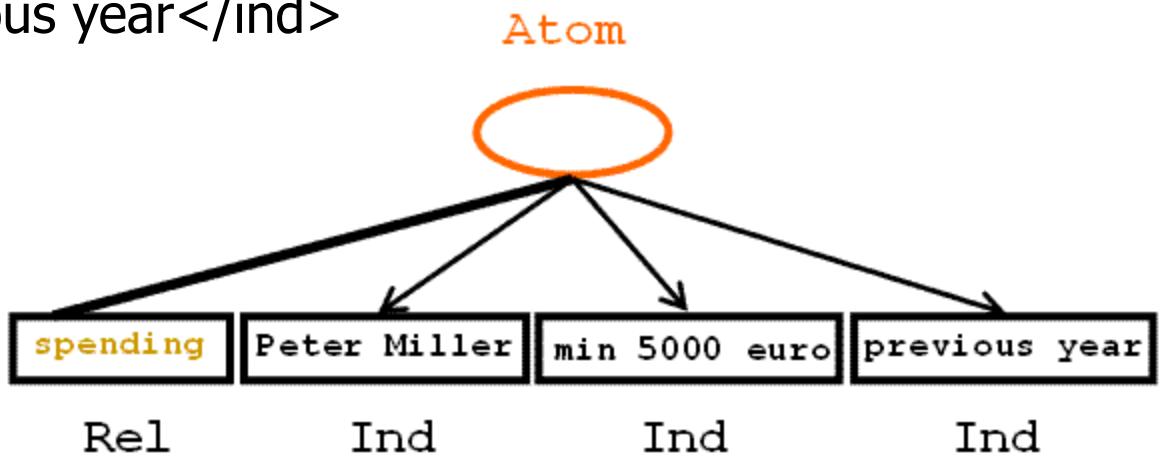
implementate prin

```
<imp>  <_head> conc </_head>
      <_body> <and> </and> </_body> </imp>
```

RuleML

Fapte

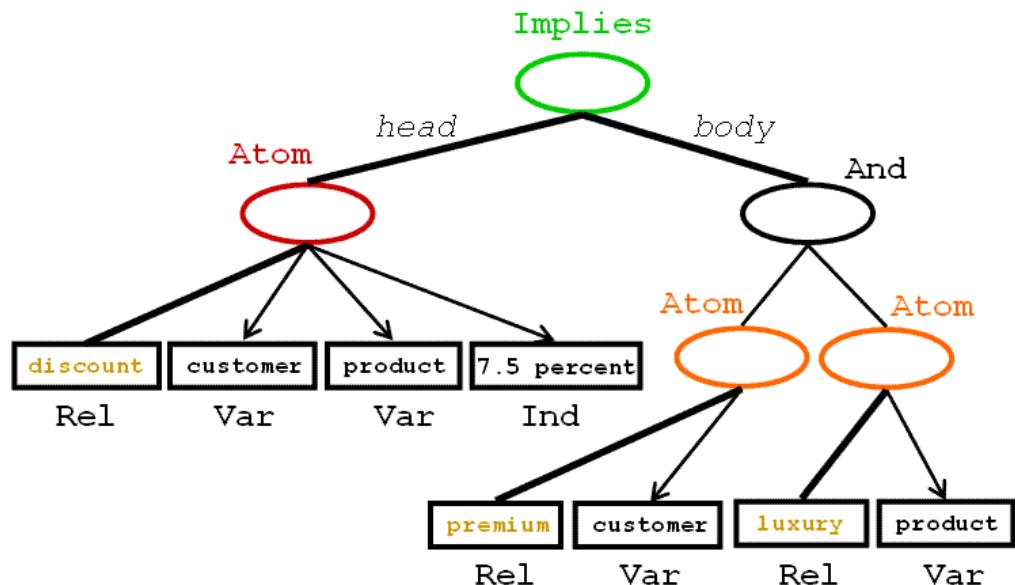
```
<atom> <rel>spending</rel>
      <ind>Peter Miller</ind>
      <ind>min 5000 euro</ind>
      <ind>previous year</ind>
</atom>
```



spending(petterMiller, min5000euro, previousYear).

Reguli de inferenta/derivare

```
<imp>
  <head> <atom>
    <rel>discount</rel>
    <var>customer</var>
    <var>product</var>
    <ind>7.5 percent</ind>
  </atom>
</head>
<body>
  <and>
    <atom>
      <rel>premium</rel>
      <var>customer</var>
    </atom>
    <atom>
      <rel>luxury</rel>
      <var>product</var>
    </atom>
  </and>
</body>
</imp>
```



discount(Customer, Product, 7.5_percent):-
 premium(Customer), luxury(Product).

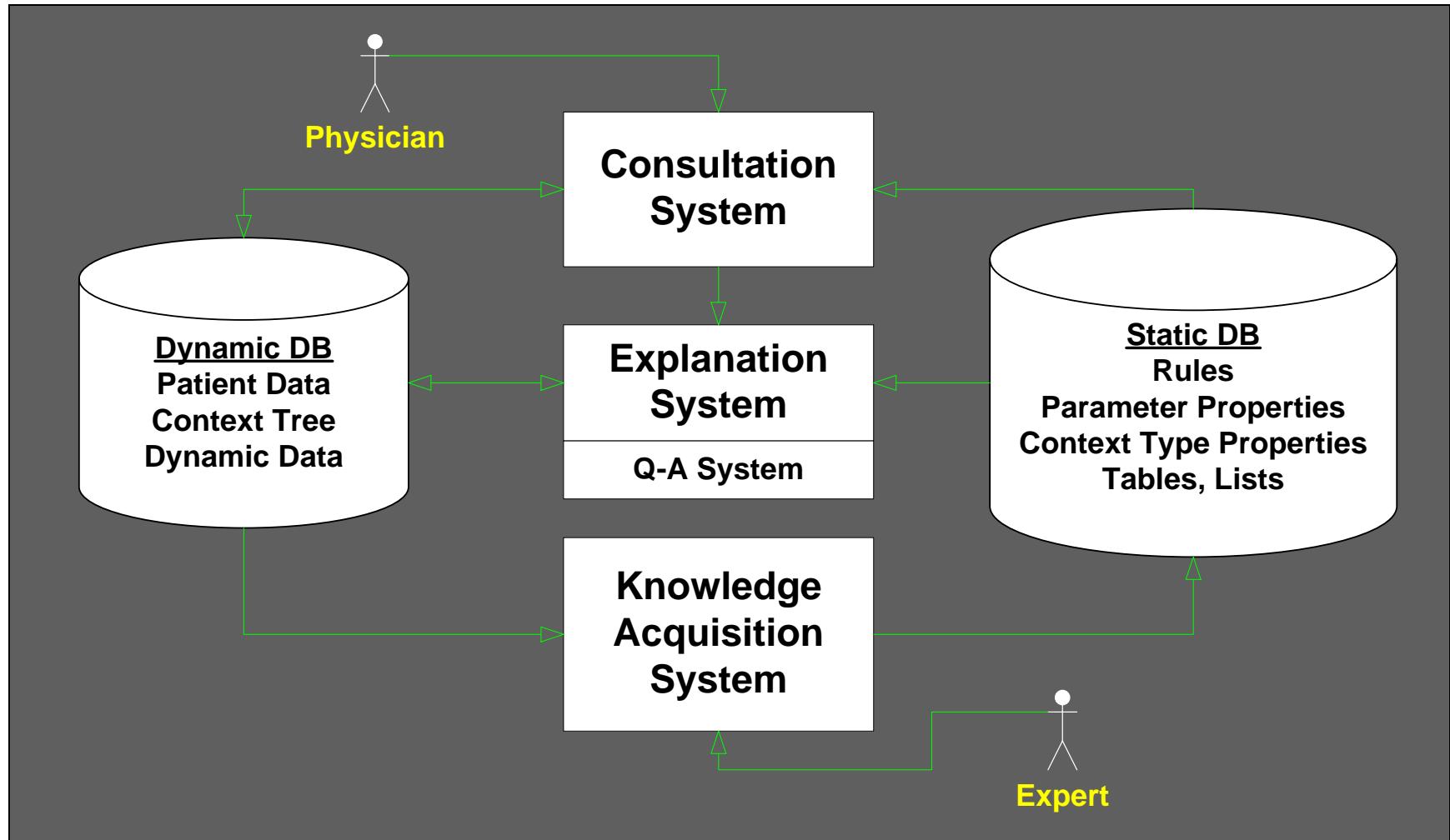
6. SBR cu inlantuire inapoi

Multe sisteme sunt combinate cu o forma de rationament incert

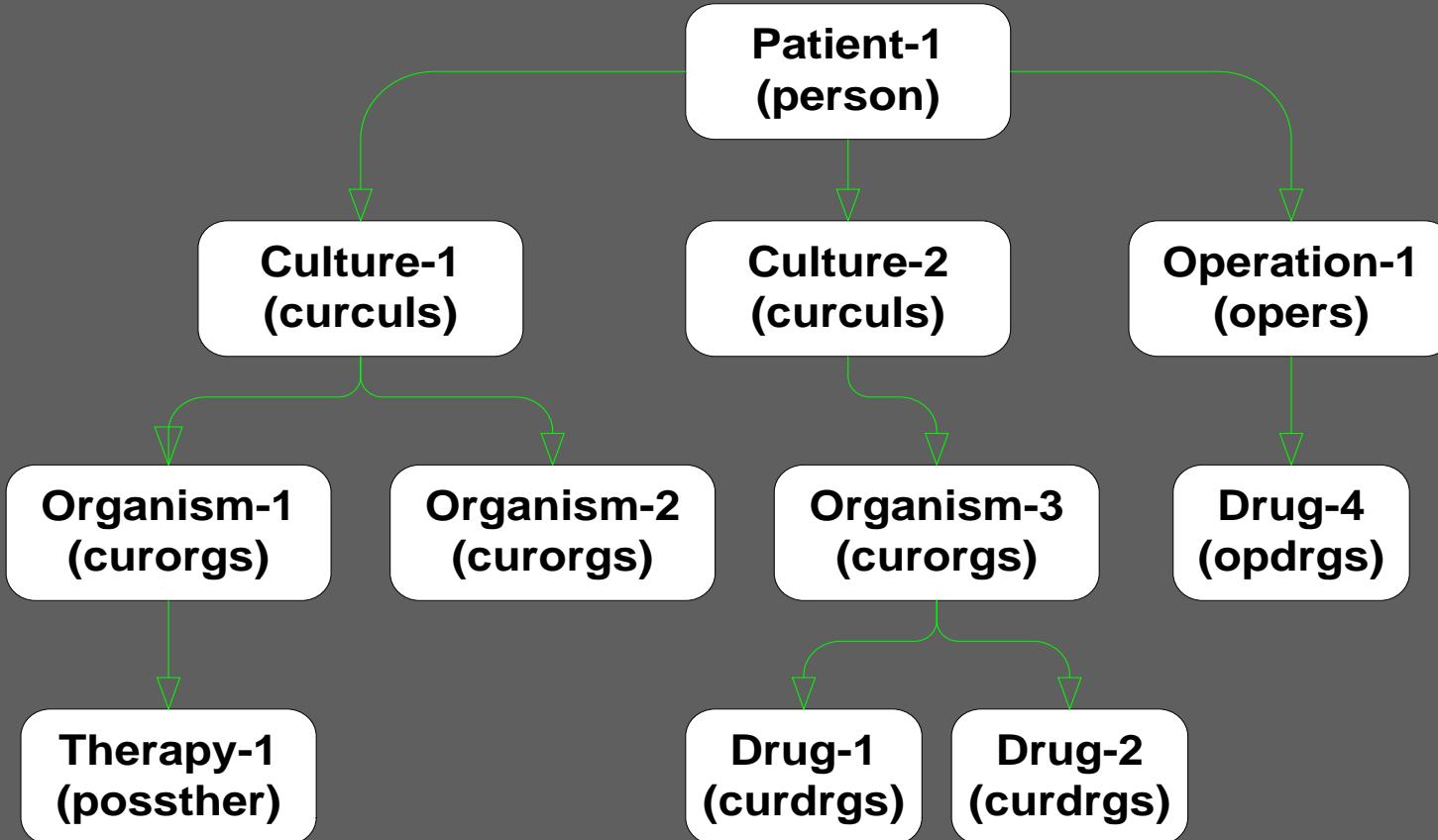
MYCIN – sistem expert pentru diagnosticarea infectiilor bacteriene ale sangelui

- Shortliffe, Davis, Buchanan, van Melle, si altii
 - Stanford Heuristic Programming Project
 - Infectious Disease Group, Stanford Medical

MYCIN - arhitectura



MYCIN – Context tree



MYCIN

Inlantuire inapoi bazat pe cautare in adancime
cu acumulare de probe

Control la nivel inalt

- Daca pacientul are o infectie semnificativa
- Identitatea organismului
- Care sunt medicamentele adecvate
- Alege pe cel mai bun

MYCIN - Explicatii

32) Was penicillinase added to this blood culture (CULTURE-1)?

****WHY**

[i.e. WHY is it important to determine whether penicillinase was added to CULTURE-1?]

[3.0] This will aid in determining whether ORGANISM-1 is a contaminant. It has already been established that

[3.1] the site of CULTURE-1 is blood, and

[3.2] the gram stain of ORGANISM-1 is grampos

Therefore, if

[3.3] penicillinase was added to this blood culture then there is weakly suggestive evidence...

7. Factori de certitudine

- Modelul MYCIN - heuristic
- Factori de certitudine / Coeficienti de incredere (**CF**)
- Model heuristic al reprezentarii cunoștințelor incerte

MYCIN - Exemplu de utilizare CF

- Regula in sistemul MYCIN
daca tipul organismului este gram-pozitiv, si
morfologia organismului este coc, si
conformatia cresterii organismului este lant
atunci exista o incredere puternica (**0.7**)
ca identitatea organismului este streptococ.
- Exemple de fapte in sistemul MYCIN :
(identitate organism-1 pseudomonas **0.8**)
(identitate organism-2 e.coli **0.15**)
(loc cultura-2 git **1.0**)

Functii de combinare a incertitudinii

(1) Probe adunate incremental

- Aceeasi valoare de atribut, h, este obtinuta pe doua cai de deductie distincte, cu doua perechi diferite de valori pentru CF, $CF[h,s1]$ si $CF[h,s2]$
- Cele doua cai de deductie distincte, corespunzatoare probelor sau ipotezelor s1 si s2 pot fi ramuri diferite ale arborelui de cautare generat prin aplicarea regulilor sau probe indicate explicit sistemului de medic.
- $CF[h, s1\&s2] = CF[h,s1] + CF[h,s2] - CF[h,s1]*CF[h,s2]$
- (identitate organism-1 pseudomonas 0.8)
- (identitate organism-1 pseudomonas 0.7)

Functii de combinare a incertitudinii

(2) Conjunctie de ipoteze

- Se aplica pentru calculul CF asociat unei premise de regula care contine mai multe conditii

daca ($A = h_1$) si ($B = h_2$) **atunci** ... - **s**

ML: ($A \ h_1 \ CF_1$) ($B \ h_2 \ CF_2$)

- $CF[h_1 \& h_2, s] = \min(CF_1[h_1, s], CF_2[h_2, s])$
- Se generalizeaza pt mai multe conditii

Functii de combinare a incertitudinii

(3) Combinarea increderii

- O valoare incerta este dedusa pe baza unei reguli care are drept conditie de intrare alte valori incerte (deduse eventual prin aplicarea altor reguli).
- Permite calculul factorului de certitudine asociat valorii deduse pe baza aplicarii unei reguli care refera valoarea in concluzie, tinind cont de CF-ul asociat premisei regulii.
- **CF[s,e]** - increderea intr-o ipoteza **s** pe baza unor probe anterioare **e**
- **CF[h,s]** - CF in **h** in cazul in care **s** este sigura
- **CF'[h,s] = CF[h,s] * CF [s,e]**

Functii de combinare a incertitudinii

(3) Combinarea increderii – cont

daca $A = h1$ si $B = h2$ atunci $C = c1 0.7$

ML: (A h1 0.9) (B h2 0.6)

$$CF(\text{premisa}) = \min(0.9, 0.6) = 0.6$$

$$\text{CF (concluzie)} = \text{CF(premisa)} * \text{CF(regula)} = 0.6 * 0.7$$

$$\text{ML: (C c1 0.42)}$$

Exemplu SBR cu inlantuire inapoi si calcul incert (CF)

- Exemplu: o baza de cunostinte pentru alegerea vinului adecvat unui meniu
- Valoarea fiecarui atribut este memorata impreuna cu coeficientul de certitudine asociat.
- Coeficientii de certitudine sunt valori pozitive in intervalul [0,1].

(vin chardonney 0.8 riesling 0.6)
- O regula poate avea asociat un coeficient de certitudine
daca sos-meniu = sos-alb
atunci culoare-vin = alba 0.6

R11:daca componenta-meniu = curcan
atunci culoare-vin = rosie 0.7
si culoare-vin = alba 0.2

R12:daca componenta-meniu = peste
atunci culoare-vin = alba

R13:daca sos-meniu = sos-alb
atunci culoare-vin = alba 0.6

R14:daca componenta-meniu = porc
atunci culoare-vin = rosie

R21:daca sos-meniu = sos-alb
atunci tip-vin = sec 0.8
si tip-vin = demisec 0.6

R22:daca sos-meniu = sos-tomat
atunci tip-vin = dulce 0.8
si tip-vin = demisec 0.5

R23:daca sos-meniu = necunoscut
atunci tip-vin = demisec

R24:daca componenta-meniu = curcan
atunci tip-vin = dulce 0.6
si tip-vin = demisec 0.4

R31:daca culoare-vin = rosie

si tip-vin = dulce

atunci vin = gamay

R32:daca culoare-vin = rosie

si tip-vin = sec

atunci vin = cabernet-sauvignon

R33:daca culoare-vin = rosie

si tip-vin = demisec

atunci vin = pinot-noir

R34:daca culoare-vin = alba

si tip-vin = dulce

atunci vin = chenin-blanc

R35:daca culoare-vin = alba

si tip-vin = sec

atunci vin = chardonnay

R36:daca culoare-vin = alba

si tip-vin = demisec

atunci vin = riesling

2.3 Limitari ale modelului CF

- Modelul coeficientilor de certitudine din MYCIN presupune ca ipotezele sustinute de probe sunt independente.
- Un exemplu care arata ce se intimpla in cazul in care aceasta conditie este violata.

Fie urmatoarele fapte:

- A: Aspersorul a functionat noaptea trecuta.
U: Iarba este uda dimineata.
P: Noaptea trecuta a plouat.

Limitari ale modelului CF - cont

A: Aspersorul a functionat noaptea trecuta.

U: Iarba este uda dimineata.

P: Noaptea trecuta a plouat.

si urmatoarele doua reguli care leaga intre ele aceste fapte:

R1: **daca** aspersorul a functionat noaptea trecuta
atunci exista o incredere puternica (0.9) ca iarba este
uda dimineata

R2: **daca** iarba este uda dimineata
atunci exista o incredere puternica (0.8) ca noaptea
trecuta a plouat

Limitari ale modelului CF - cont

- $CF[U,A] = 0.9$
- deci proba aspersor sustine iarba uda cu 0.9

- $CF[P,U] = 0.8$
- deci iarba uda sustine ploaie cu 0.8

- $CF[P,A] = 0.8 * 0.9 = 0.72$
- deci aspersorul sustine ploaia cu 0.72
- **Solutii**

Inteligentă Artificială

Universitatea Politehnica Bucuresti
Anul universitar 2020-2021

Adina Magda Florea



Curs nr. 7

Planificare automata

- PA – caracteristici
- PA - reprezentare
- Planificare liniara in sistemul STRIPS
- Grafuri de planificare
- Planificare neliniara – TWEAK
- Planificare ierarhica si contingenta

1. Planificare automata - caracteristici

Descompunerea problemelor in subprobleme

Tipuri de planificare:

- liniara
- neliniara
- ierarhica
- contingenta

Rationament de bun simt:

- Problema cadrului, problema calificarii si problema ramificarii

Algoritmi de planificare

- **Algoritmi de planificare discreta** – spațiul posibil al stărilor de planificare este discret – trebuie să fie numarabil; în cele mai multe cazuri este finit
- **Algoritmi de planificare continuă** – spațiul de mișcări este continuu
- Pot fi în condiții certe (mediu determinist) sau incerte (mediu nedeterminist)

2. Planificare automata - reprezentare

A) Reprezentare prin spațiul stărilor

- Spațiul stărilor S – finit
- Pentru fiecare stare s , mulțimea acțiunilor aplicabile în acea starea – $u = U(s)$

Mulțimea de acțiuni posibile din toate stările

$$U = \bigcup_{s \in S} U(s)$$

- Funcția de tranziție între stări $f: S \times U(S) \rightarrow S$
 $s' = f(s, u)$
- Starea inițială $s_I \in S$
- $S_G \subset S$ – mulțimea de stări scop

Algoritmi de planificare in spațiul stăriilor

- Cătarea pe nivel
- Căutare în adâncime
- Iterative deepening
- Algoritmul lui Dijkstra – plan optim (daca putem expanda integral spatial)
- A* – plan optim
- Variante backward ale algoritmilor
- Căutare bidirecțională
- Iterarea valorii pentru planuri optime (programare dinamica) – forward si backward

B) Reprezentare bazată pe logică simbolică

- Folosirea unei reprezentări bazate pe logica simbolica
- Reprezintă problema de planificare foarte compact
- Ieșire convenabilă
- Permite specificarea implicită a problemei cadrului
- STRIPS-like representations

Reprezentare bazată pe logică simbolică

- O mulțime finită de predicate **P**
- O mulțime finită de operatori de plan **O** cu
 - Precondiții – literali pozitivi sau negativi
 - Postcondiții (efect) - literali pozitivi sau negativi
- O mulțime inițială de literali **S** care descrie starea inițială – numai literali pozitivi
- O mulțime de literali **G** care descrie starea scop – literali pozitivi sau negativi

Exemplul STRIPS și TWEAK

Algoritmi de planificare în reprezentare logică

- **Regresia scopului** (planificare backward) – de ex. algoritm STRIPS – potrivit pentru probleme de planificare liniară
- **Planificare cu planuri parțiale** – de ex. TWEAK – potrivit pentru probleme de planificare neliniară
- **Algoritm cu grafuri de planificare** – utilizează o structură de date care reprezintă compact informații despre stările la care se poate ajunge

C) Planificarea ca o problemă de realizabilitate a unei formule logice

- Transformă problema de planificare într-o enormă problemă de realizabilitate booleană
- Problema realizabilitatii boleene (SATISFIABILITY or SAT)
- Problema NP-completea (teorema Cook–Levin)
- Reprezinta starea initiala, starea scop, operatorii, axiomele cadru si axiomele de excludere complete (un singur operator la fiecare pas)
- Determina daca există o atribuire de **a,f** variabilelor din problemă pentru care formula este adevărată

Planificarea ca o problemă de realizabilitate a unei formule logice

- Algoritm clasic Davis-Putnam pentru realizabilitate – căutare în adâncime cu atribuirea iterativă a unor asignări și revine dacă o asignare face formula falsă
- Algoritmi de căutare locală
 - GSAT, WalkSAT
- Recent *algoritmii SAT euristici* pot rezolva probleme cu zeci de mii de variabile (formulele pot avea de ordinul a milioane de simboluri)

3. Planificare STRIPS

■ Reprezentare

- Reprezentarea starilor cautarii
- Operatori de plan
- Predicate
- Axiome

■ Cautarea solutiei

- Forward planning/search
- Backward planning / regression

Planificare STRIPS

- Actiuni (operatori de plan) cu preconditii si postconditii
- Cautare inainte in spatiul starilor
- Cautare inapoi (regresie) in spatiul starilor

PA Forward

- Algoritm Forward(S ,Scopuri, A , Cale)

1. **daca** S staisface scopurile din Scopuri

atunci intoarce Cale

altfel

- 1.1 Act=alege din A o actiune cu precond satisfacute de S

- 1.2 **daca** nu exista A **atunci** intoarce *Fail*

- 1.3 **altfel** fie S' =efectul simularii executiei Act in S

intoarce

Forward(S' ,Scopuri, A ,conc(Cale, A))

sfarsit

PA Backward

- Algoritm Regresie(S,Scopuri,A, Cale)

1. **daca** S staisface scopurile din Scopuri

atunci intoarce Cale

altfel

- 1.1 Act=alege din A o actiune cu efectul satisfacut in S

- 1.2 G=regresia scopurilor prin A

- 1.3 **daca** nu exista A **sau** G este nedefinit **sau** G include Scopuri **atunci** intoarce *Fail*

- 1.4 **altfel** intoarce

Regresie(G,Scopuri,A,conc(A,Cale))

sfarsit

Planificare liniara in sistemul STRIPS

■ Operatori de plan

- *Actiune* care reprezinta actiunea asociata operatorului.
- *Lista Preconditiilor* ce contine formulele care trebuie sa fie adevarate intr-o stare a problemei pentru ca operatorul sa poata fi aplicat - LP.
- *Lista Adaugarilor* ce contine formulele care vor deveni adevarate dupa aplicarea operatorului - LA.
- *Lista Eliminarilor* ce contine formulele care vor deveni false dupa aplicarea operatorului - LE.

Reprezentarea STRIPS

- Operatori de plan

$STACK(x,y)$, $UNSTACK(x,y)$, $PICKUP(x)$, $PUTDOWN(x)$

- Predicate:

$ON(x,y)$, $ONTABLE(x)$, $CLEAR(x)$, $HOLD(x)$, $ARMEMPTY$

- Axiome:

$(\exists x) (HOLD(x)) \rightarrow \sim ARMEMPTY$

$(\forall x) (ONTABLE(x) \rightarrow \sim (\exists y) (ON(x,y)))$

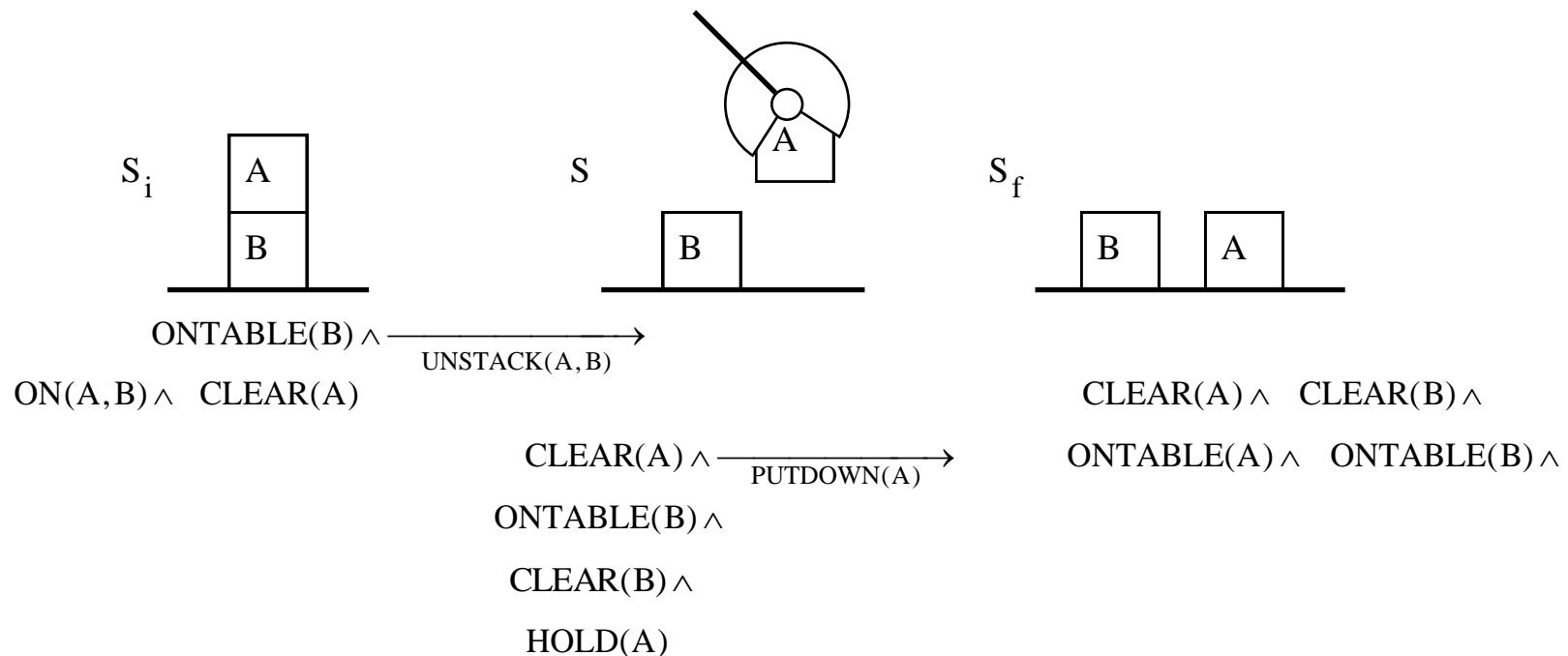
$(\forall x) (\sim (\exists y) (ON(y,x)) \rightarrow CLEAR(x))$

Reprezentarea STRIPS - cont

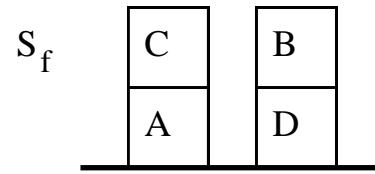
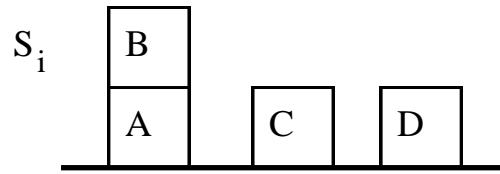
■ Operatori de plan

| | | |
|---------------|-----|--|
| STACK(x,y) | LP: | CLEAR(y) \wedge HOLD(x) |
| | LE: | CLEAR(y) \wedge HOLD(x) |
| | LA: | ON(x,y) \wedge ARMEMPTY |
| UNSTACK (x,y) | LP: | ON(x,y) \wedge CLEAR(x) \wedge ARMEMPTY |
| | LE: | ON(x,y) \wedge ARMEMPTY |
| | LA: | HOLD(x) \wedge CLEAR(y) |
| PICKUP(x) | LP: | CLEAR(x) \wedge ONTABLE(x) \wedge ARMEMPTY |
| | LE: | ONTABLE(x) \wedge ARMEMPTY |
| | LA: | HOLD(x) |
| PUTDOWN (x) | LP: | HOLD(x) |
| | LE: | HOLD(x) |
| | LA: | ONTABLE(x) \wedge ARMEMPTY |

Executia planului



Functionare STRIPS



$ON(B, A) \wedge$
 $ONTABLE(A) \wedge$
 $ONTABLE(C) \wedge$
 $ONTABLE(D) \wedge$
ARMEMPTY

$ON(C, A) \wedge$
 $ON(B, D) \wedge$
 $ONTABLE(A) \wedge$
ONTABLE(D)

Stiva 1

$ON(C, A)$
 $ON(B, D)$
 $ON(C, A) \wedge ON(B, D) \wedge OTAD$

Stiva 2

$ON(B, D)$
 $ON(C, A)$
 $ON(C, A) \wedge ON(B, D) \wedge OTAD$

Functionare STRIPS - cont

/* pentru realizarea scopului ON(C,A) */

STACK(C,A)
ON(B,D)
ON(C,A) \wedge ON(B,D) \wedge OTAD

/* preconditiile operatorului STAC(C,A)*/

CLEAR(A)
HOLD(C)
CLEAR(A) \wedge HOLD(C)
STACK(C,A)
ON(B,D)
ON(C,A) \wedge ON(B,D) \wedge OTAD



$S_1 \rightarrow S_2$: ONTABLE(A) \wedge ONTABLE(C) \wedge ONTABLE(D) \wedge ON(B,D) \wedge ARMEMPTY

Plan = (UNSTACK(B,A),STACK(B,D))

S_4 : ONTABLE(A) \wedge ONTABLE(D) \wedge ON(B,D) \wedge ON(C,A) \wedge ARMEMPTY

Plan = (UNSTACK(B,A),STACK(B,D),PICKUP(C),STACK(C,A))

Algoritm STRIPS

- Variabila **S** - memoreaza descrierea starii curente a universului problemei;
- **Stiva** - memoreaza stiva de scopuri satisfacute pe calea curenta de cautare;
- **Scopuri** - pastreaza lista scopurilor nesatisfacute pe calea curenta;
- Structura **Operator** avand campurile: *Actiune*, *Preconditii*, *ListaAdaugari* si *ListaEliminari*
(Operator.Preconditii)

Algoritm STRIPS

Algoritm: Planificare liniara in STRIPS

SatisfacereScopuri (Scopuri, S, Stiva)

1. **pentru** fiecare Scop din Scopuri **executa**
 - 1.1. StareNoua \leftarrow RealizeazaScop(Scop, S, Stiva)
 - 1.2. **daca** StareNoua = INSUCCES
atunci intoarce INSUCCES
2. **daca** toate scopurile din Scopuri sunt satisfacute in starea StareNoua
atunci intoarce StareNoua
3. **altfel** intoarce INSUCCES
sfarsit.

SatisfacereScopuri \rightarrow RealizeazaScop
RealizeazaScop \rightarrow AplicaOperator
AplicaOperator \rightarrow SatisfacereScopuri

Algoritm STRIPS

RealizeazaScop (Scop, S, Stiva)

1. **daca** Scop este marcat satisfacut in starea S
atunci intoarce S
2. **daca** Scop apartine Stiva
atunci intoarce INSUCCES
3. OperatoriValizi $\leftarrow \{O \mid O \text{ poate satisface scopul Scop}\}$
4. **pentru** fiecare operator O din OperatoriValizi **executa**
 - 4.1. StareNoua \leftarrow **AplicaOperator**(O, S, Stiva $\cup \{Scop\}$)
 - 4.2. **daca** StareNoua \leftrightarrow INSUCCES
atunci
 - 4.2.1. Marcheaza scopul Scop satisfacut in starea StareNoua
 - 4.2.2. **intoarce** StareNoua
5. **intoarce** INSUCCES
sfarsit.

Algoritm STRIPS

AplicaOperator (Operator, Stare, Stiva)

1. $\text{StareNoua} \leftarrow \text{SatisfacereScopuri}(\text{Operator.Preconditii}, \text{Stare}, \text{Stiva})$
2. **daca** $\text{StareNoua} \leftrightarrow \text{INSUCCES}$

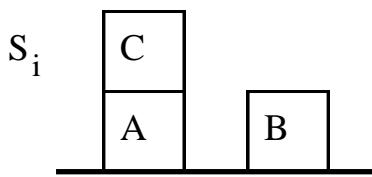
atunci

- 2.1. adauga Operator.Actiune la Plan
- 2.2. $\text{StareNoua} \leftarrow \text{StareNoua} - \text{Operator.ListaEliminari}$
- 2.3. **intoarce** $\text{StareNoua} \cup \text{Operator.ListaAdaugari}$

3. **altfel intoarce** INSUCCES

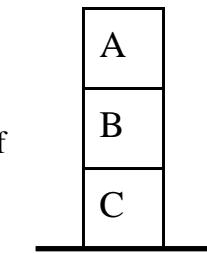
sfarsit.

Anomalia lui Sussman



$\text{ON}(C, A) \wedge$
 $\text{ONTABLE}(A) \wedge$
 $\text{ONTABLE}(B) \wedge$
 ARMEMPTY

Stiva 1



$\text{ON}(A, B) \wedge$
 $\text{ON}(B, C)$

Stiva 2

~ $\text{ON}(A, B)$

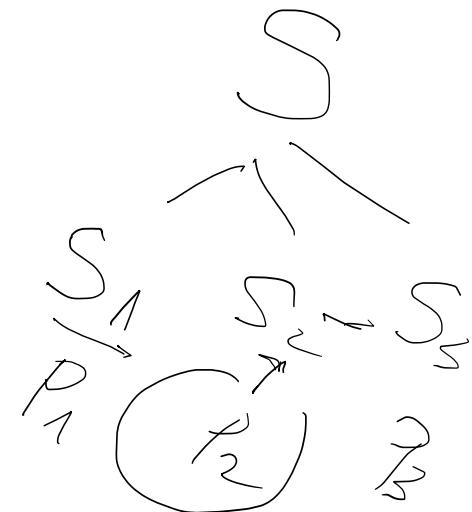
$\text{ON}(B, C)$

$\text{ON}(A, B) \wedge \text{ON}(B, C)$

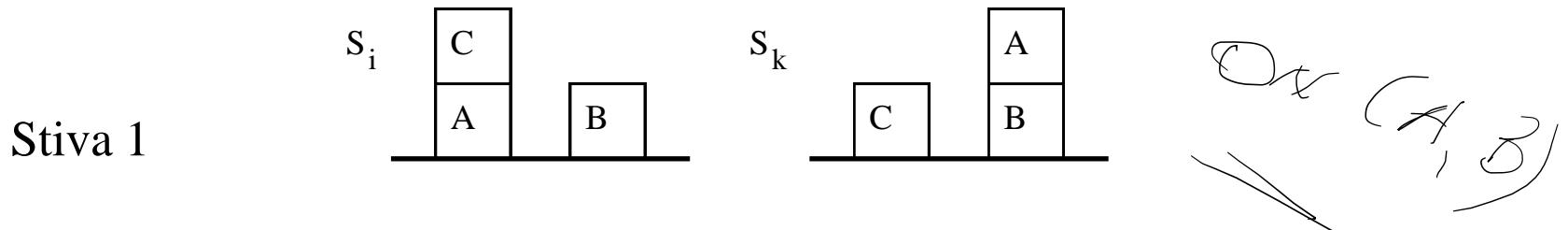
$\text{ON}(B, C)$

$\text{ON}(A, B)$

$\text{ON}(A, B) \wedge \text{ON}(B, C)$

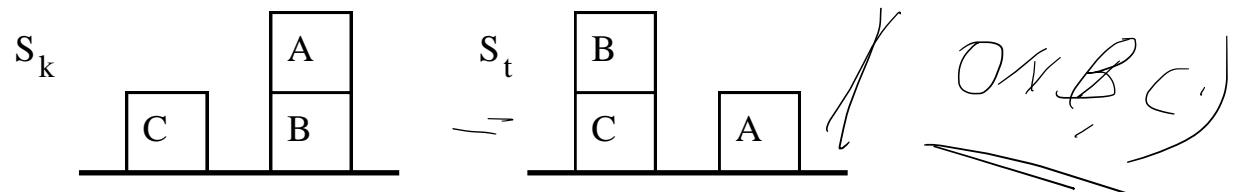


Anomalia lui Sussman



$S_i \rightarrow S_k: \text{ONTABLE}(B) \wedge \text{ON}(A, B) \wedge \text{ONTABLE}(C) \wedge \text{ARMEMPTY}$

$$\text{Plan}_{S_i \rightarrow S_k} = (\text{UNSTACK}(C, A), \text{PUTDOWN}(C), \text{PICKUP}(A), \text{STACK}(A, B))$$



$S_k \rightarrow S_t: ON(B,C) \wedge ONTABLE(A) \wedge ONTABLE(C) \wedge ARMEMPTY$

$\text{Plan}_{S_t \rightarrow S_t} = (\text{UNSTACK}(A, B), \text{PUTDOWN}(A), \text{PICKUP}(B), \text{STACK}(B, C))$

$S_t \rightarrow S_f: ON(A,B) \wedge ON(B,C)$

$\text{Plan}_{S_t \rightarrow S_f} = (\text{PICKUP}(A), \text{STACK}(A, B))$

// Plan = (UNSTACK(C,A), PUTDOWN(C), PICKUP(A), STACK(A,B), UNSTACK(A,B)
 PUTDOWN(A), PICKUP(B), STACK(B,C), PICKUP(A), STACK(A,B))

4. Grafuri de planificare

- Se aplica operatorilor instantiati (logica propozitiilor)
- Un graf de planificare este un **graf orientat organizat pe niveluri**:
 - nivel S_0 (S_i) pt starea S_0 – noduri care reprezinta literali adevarati in S_0 (S_i)
 - nivel A_0 (A_i) – noduri care reprezinta actiuni ce pot fi aplicate in S_0 (S_i)
- S_i – toti literalii care pot fi adevarati la momentul i in fct de actiunile execute anterior
- A_i – toate actiunile care pot fi execute deoarece au preconditiile satisfacute

Restrictii in GP

- Se adauga si actiuni de persistenta – **no-op**
- Conflicte intre actiuni ce nu pot fi executate impreuna – excludere mutuala – **legaturi mutex actiuni**
- Conflicte intre literari - – **legaturi mutex lietrali**
- **Graf stabilizat** (leveled off)
- A_i – toate actiunile in S_i + restrictii
- S_i – toti literalii adevarati pt orice alegere posibila de actiuni pe nivelul A_{i-1} + restrictii

Restrictii in GP

- **Legaturi (restrictii) mutex intre actiuni:**
 - **postconditii inconsistente (PI)** – o actiune neaga postconditia alteia
 - **destructivitate/interferente (D)** – postconditia unei actiuni este negarea preconditiei alteia
 - **necesitati competitive (NC)** – preconditia unei actiuni este mutual exclusiva cu preconditia alteia
- **Legaturi (restrictii) mutex intre literali:**
 - un literal si acelasi literal negat
 - fiecare pereche de actiuni care pot adauga cei 2 literali sunt mutex

Exemplu

- Exemplu (Credit: S. Russel & P. Norvig: Artificial Intelligence: A Modern Approach, Prentice Hall, 2009)

Actiune **Eat(Cake)**

Preconditii: Have(Cake)

Postconditii: \neg Have(Cake) \wedge Eaten(Cake)

Actiune **Bake(Cake)**

Preconditii: \neg Have(Cake)

Postconditii: Have(Cake)

S_i

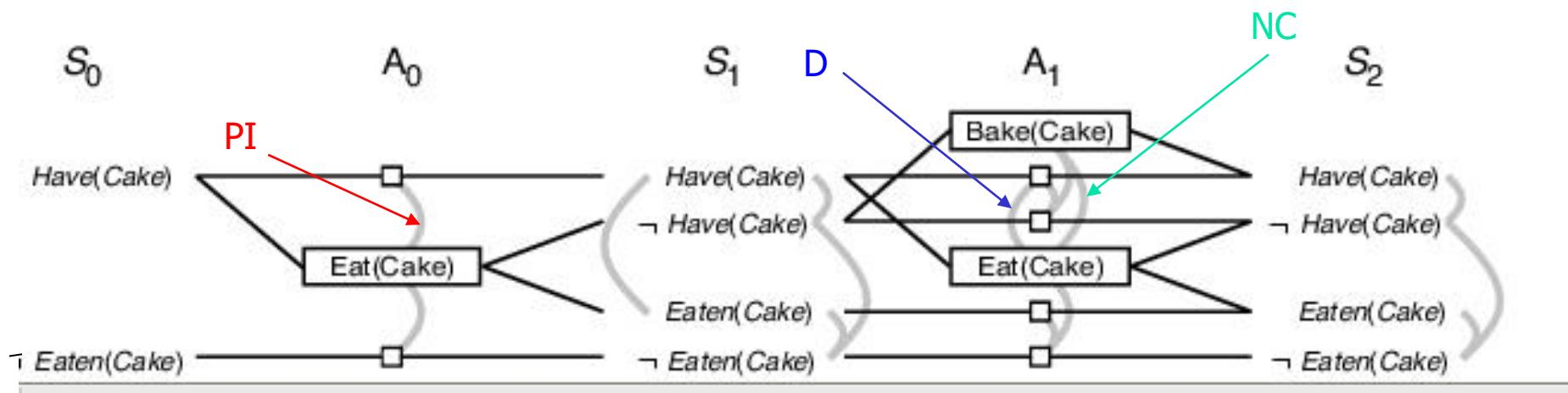
Have(Cake)

\neg Eaten(Cake)

S_f

Have(Cake) \wedge Eaten(Cake)

Exemplu



- Figure Credit: S. Russel & P. Norvig: Artificial Intelligence: A Modern Approach, Prentice Hall, 2009

Despre GP

- Un GP este polynomial in raport cu dimensiunea problemei de planificare
- l literali si a actiuni
- S_i are max l noduri si l^2 legaturi mutex
- A_i are max $a+l$ noduri (inclusiv no-op) si $(a+l)^2$ legaturi mutex si $2(al+l)$ legaturi preconditi si efect
- Un graf cu n niveluri are o dimensiune $O(n(a+l)^2)$

Despre GP

- Folosit si pt **estimari euristice**
- Costul necesar pt a satisface g_i din starea scop S_f
- **Costul nivel** al lui g_i
- Estimare cost **conjunctie de scopuri**
 - nivel-maxim – maxim nivel g_i
 - nivel-suma – suma niveluri g_i
 - nivel set – nivelul in care apar toate g_i a.i. intre nici o pereche sa nu existe legaturi mutex

Algoritm GP

Algoritm: Planificare cu graf de planificare

GrafPlan(problema)

1. $\text{graf} \leftarrow \text{graf_initial}(\text{problema})$
2. $\text{scopuri} \leftarrow \text{conjunctie}(\text{literali scop})$
3. $\text{nogoods} \leftarrow \text{o tabela hash vida}$
4. **pentru** $j \leftarrow 0$ la infinit **executa**
 - 4.1 **daca** $\text{scopuri ne-mutex in } S_j$ din graf **atunci**
 - $\text{solutie} \leftarrow \text{Extrage_Solutie}(\text{graf}, \text{scopuri}, \text{niv}(\text{graf}), \text{nogoods})$
 - **daca** $\text{solutie} \neq \text{esec}$ **atunci intoarce** solutie
 - 4.2 **daca** graf **si** nogoods s-au stabilizat **atunci intoarce** esec
 - 4.3 $\text{graf} \leftarrow \text{Expandeaza_Graf}(\text{graf}, \text{problema})$

sfarsit

Algoritm GP

- **nogoods** – daca Extract_Solutie nu gaseste solutie pt o multime de scopuri pe un nivel l atunci se memoreaza perechea $(l, \text{scopuri})$ ca un nogood
- **Extract_Solutie – CSP Boolean**
 - Variabile: actiuni pe fiecare nivel
 - Valori variabile: *in* sau *out* (in plan)
 - Restrictii: legaturi mutex, necesitatea satisfacerii fiecarui scop si preconditii

Algoritm GP

■ Extragă_Solutie – căutare backward

- *Fiecare stare contine un pointer la un nivel in graf si o multime de scopuri nesatisfacute*
- Stare initiala este ultimul nivel S_n din graf + scopuri din problema
- Actiunile posibil de executat intr-o stare de nivel S_i – un subset de actiuni din A_{i-1} fara mutex si care au ca postconditii scopurile din S_i – vor fi mai multe astfel de seturi de actiuni
- Starea rezultata are nivelul S_{i-1} si are ca subscopuri preconditiile actiunilor selectate
- Terminare – sa se ajunga la o stare din S_0 a.i toate scopurile sunt satisfacute

Algoritm GP

- **Extrage_Solutie**
- In cazul in care **Extrage_Solutie** nu reuseste sa gaseasca o solutie pentru un set de scopuri pe un nivel i , se inregistreaza perechea (nivel,scopuri) ca o pereche **no-good**.
- Cand se apeleaza Extrage_Solutie din nou pentru acelasi nivel si aceleasi scopuri se gaseste inregistrarea **no-good** si insucces.
- Construirea grafului – polinomiala
- Extrage solutie NU
- **Euristici**
 - alege literalul cu costul nivel cel mai mare
 - pentru a satisface acest literal alege actiuni a.i. suma (sau maximul) costurilor nivel ale preconditiilor sa fie minima

Exemplu

- (Credit: S. Russel & P. Norvig: Artificial Intelligence: A Modern Approach, Prentice Hall, 2009)

Actiune **Remove(obj,loc)**

Preconditii: $\text{At}(\text{obj}, \text{loc})$

Postconditii: $\neg \text{At}(\text{obj}, \text{loc}) \wedge \text{At}(\text{obj}, \text{Ground})$

Actiune **PutOn(t,Axle)**

Preconditii: $\text{Tire}(t) \wedge \text{At}(t, \text{Ground}) \wedge \neg \text{At}(\text{Flat}, \text{Axle})$

Postconditii: $\neg \text{At}(t, \text{Ground}) \wedge \text{At}(t, \text{Axle})$

Actiune **LeaveOvernight**

Preconditii:

Postconditii: $\neg \text{At}(\text{Spare}, \text{Ground}) \wedge \neg \text{At}(\text{Spare}, \text{Axle}) \wedge \neg \text{At}(\text{Spare}, \text{Trunk})$
 $\wedge \neg \text{At}(\text{Flat}, \text{Ground}) \wedge \neg \text{At}(\text{Flat}, \text{Axle}) \wedge \neg \text{At}(\text{Flat}, \text{Trunk})$

S_i

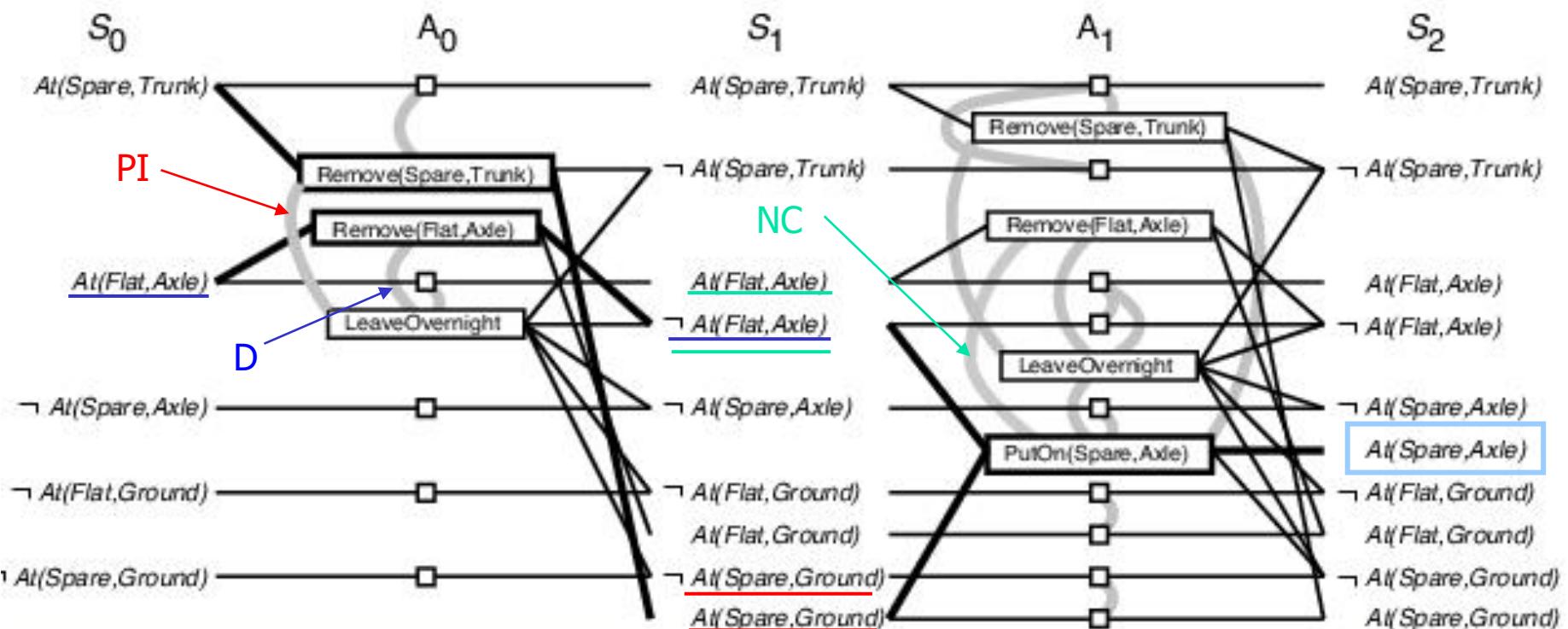
$\text{Tire}(\text{Flat}), \text{Tire}(\text{Spare})$

$\text{At}(\text{Flat}, \text{Axle}), \text{At}(\text{Spare}, \text{Trunk})$

S_f

$\text{At}(\text{Spare}, \text{Axle})$

Exemplu



- Picture Credit: S. Russel & P. Norvig: Artificial Intelligence: A Modern Approach, Prentice Hall, 2009

5. Planificare neliniara - TWEAK

- Inregistrarea restrictiilor
(temporale, unificare/codesemnare)
- Nivelul de reprezentare a planului;
- Nivelul de modificare a planului pentru a obtine un plan care realizeaza scopul problemei;

Reprezentarea TWEAK

| | |
|---------------|--|
| Actiune | STACK(x,y) |
| Preconditii: | CLEAR(y) \wedge HOLD(x) |
| Postconditii: | ARMEMPTY \wedge ON(x,y) \wedge \sim CLEAR(y) \wedge \sim HOLD(x) |

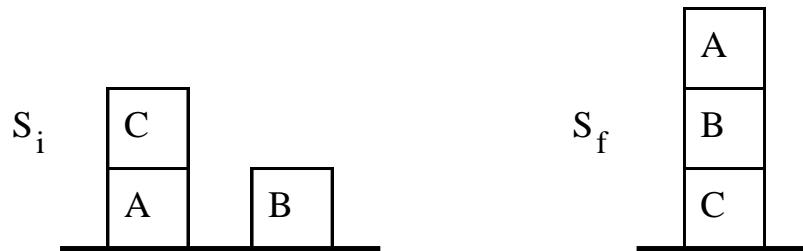
| | |
|---------------|---|
| Actiune: | PICKUP(x) |
| Preconditii: | CLEAR(x) \wedge ONTABLE(x) \wedge ARMEMPTY |
| Postconditii: | HOLD(x) \wedge \sim ONTABLE(x) \wedge \sim ARMEMPTY |

Sinteza planului

Operatii de modificare a planului:

- (1) *adaugarea de pasi* este operatia prin care se creaza noi pasi care se adauga la plan;
- (2) *promovarea* este operatia de stabilire a unei ordonari (temporale) intre doi pasi de plan;
- (3) *legarea simpla* este operatia de atribuire de valori variabilelor pentru a valida preconditiile unui pas de plan;
- (4) *separarea* este operatia de impiedicare a atribuirii anumitor valori unei variabile;
- (5) *eliminarea destructivitatii* este operatia de introducere a unui pas S3 (un pas deja existent in plan sau un pas nou) intre pasii S1 si S2, in scopul de a adauga un fapt invalidat de pasul S1 si necesar in pasul S2 (S1 amenintare pt S2).

Sinteza planului



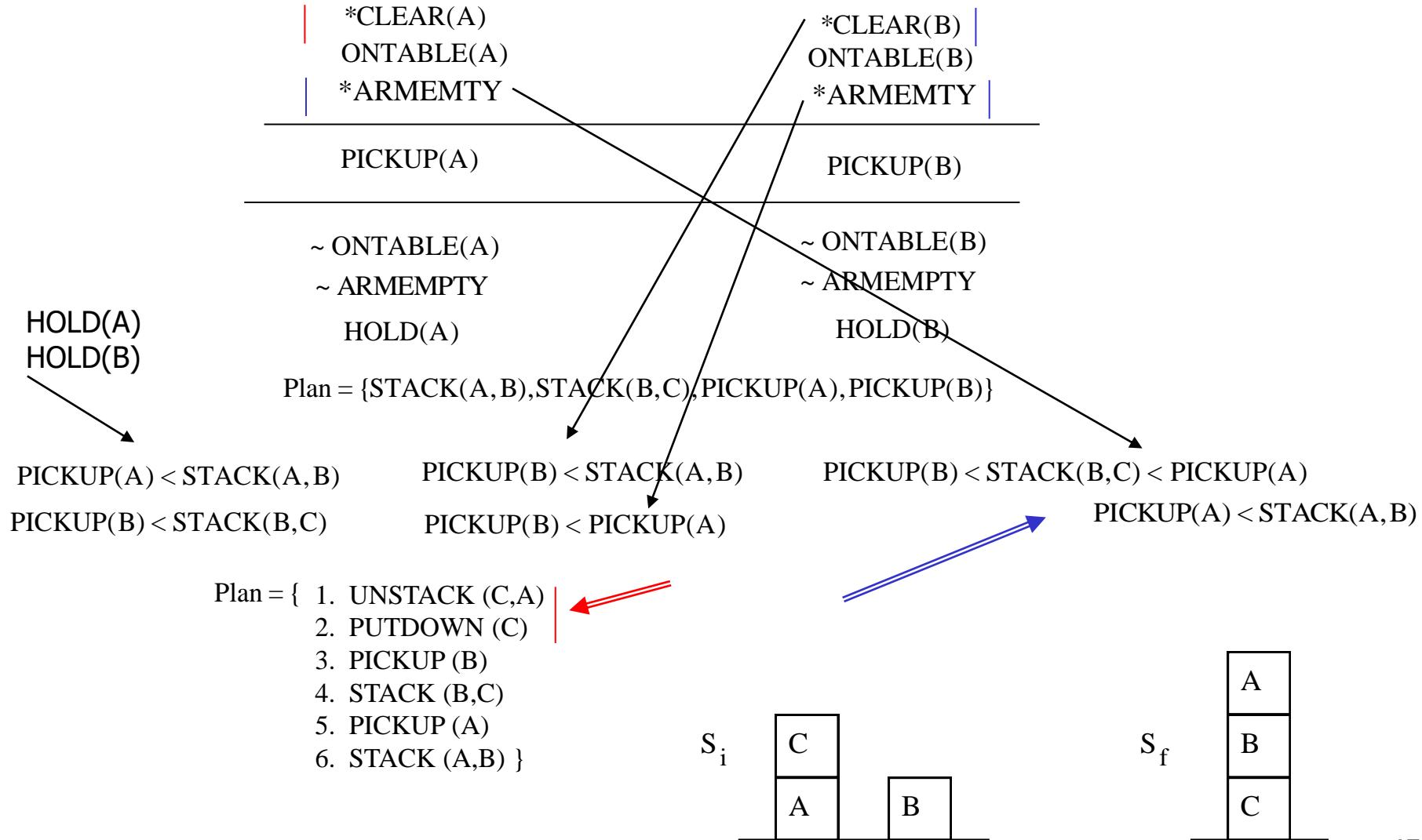
$\text{ON}(C, A) \wedge$
 $\text{ONTABLE}(A) \wedge$
 $\text{ONTABLE}(B) \wedge$
 ARMEMPTY

$\text{ON}(A, B) \wedge$
 $\text{ON}(B, C)$

| | | |
|----------------------|----------------------|--------------------------------|
| CLEAR(B) *HOLD(A) | CLEAR(C) *HOLD(B) | Plan = {STACK(A,B),STACK(B,C)} |
| STACK(A,B) | STACK(B,C) | |

| | |
|----------------------------------|---|
| ON(A,B) ~CLEAR(B) ~HOLD(A) | ON(B,C) ~CLEAR(C) ~HOLD(B) ARMEMTY |
|----------------------------------|---|

Sinteza planului



Algoritm TWEAK

Algoritm: Planificare neliniara in TWEAK

1. Initializeaza Plan $\leftarrow \{ \}$
2. Initializeaza S cu multimea formulelor care definesc starea scop
3. **cat timp** $S \neq \{ \}$ **executa**
 - 3.1. Alege si elimina o formula F din S
 - 3.2. **daca** F nu este satisfacuta in starea curenta
atunci
 - 3.2.1. Alege o operatie de modificare a planului
 - 3.2.2. Aplica operatia si adauga efectul ei in Plan

Algoritm TWEAK

- 3.3. Verifica pentru toti pasii din Plan satisfacerea preconditiilor
- 3.4. **pentru** fiecare preconditie nesatisfacuta a unui pas din Plan **executa**
 Adauga preconditia la S
4. Genereaza ordinea totala a elementelor din Plan pe baza relatiilor de ordine individuale
5. **daca** planul Plan este partial
atunci
 - 5.1. Instantiaza variabilele planului
 - 5.2. Transforma arbitrar ordinea partiala in ordine totala**sfarsit.**

Modelul formal al planificarii TWEK

- O formula este *adevarata* intr-o stare daca unifica cu o formula care face parte din stare respectiva.
- Un pas de plan *afirma* o formula in starea sa de iesire daca formula unifica cu o postconditie a pasului.
- Un pas de plan *infirma* o formula in starea sa de iesire daca afirma negarea acelei formule.
- Un pas de plan poate fi executat numai daca toate preconditiile sale sunt adevarate in starea sa de intrare.
- Starea de iesire a pasului de plan este starea de intrare din care se sterg formulele infirmate de catre pasul de plan si se adauga formulele afirmate de pasul de plan.
- Ordinea de efectuare a operatiilor de stergere si adaugare este importanta.

Modelul formal al planificarii TWEAK

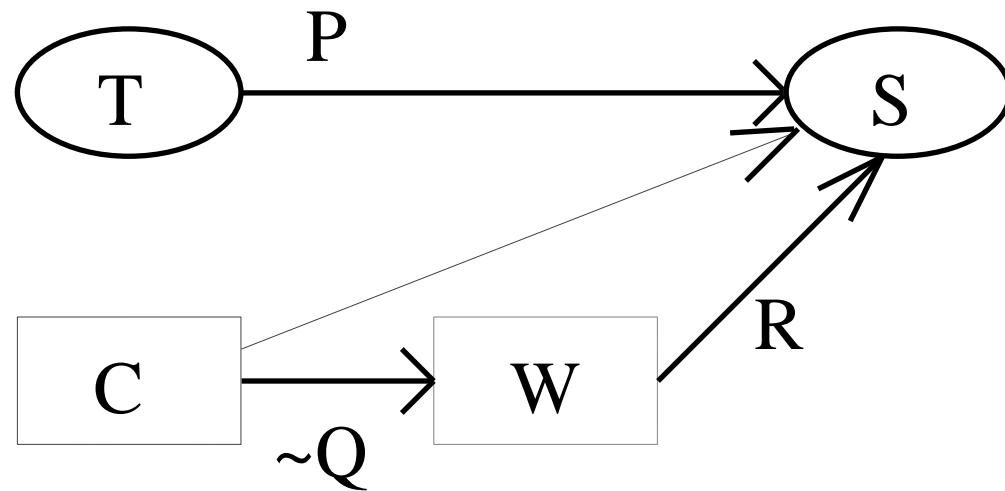
Criteriul adevarului necesar

O formula **P** este necesar adevarata intr-o stare **S** daca si numai daca se indeplinesc urmatoarele doua conditii:

- (1) exista o stare **T** egala cu/sau necesar anterioara lui **S** in care **P** este necesar afirmata (adaugata);
- (2) pentru fiecare pas **C** posibil de executat inaintea lui **S** si pentru fiecare formula **Q** care poate unifica cu **P** pe care **C** o infirma, exista un pas **W** necesar intre **C** si **S** care afirma **R**, **R** fiind o formula pentru care **R** si **P** unifica ori de cate ori **P** si **Q** unifica.

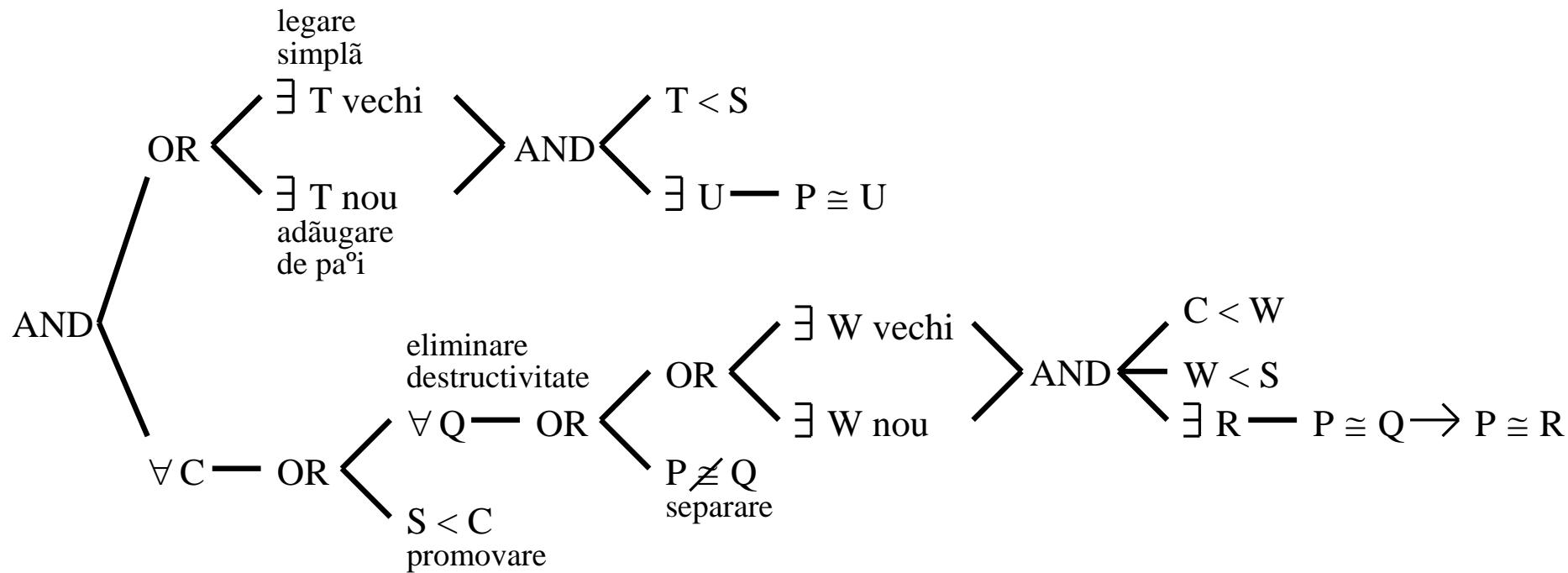
Modelul formal al planificarii TWEAK

Criteriul adevarului necesar



Modelul formal al planificarii TWEAK

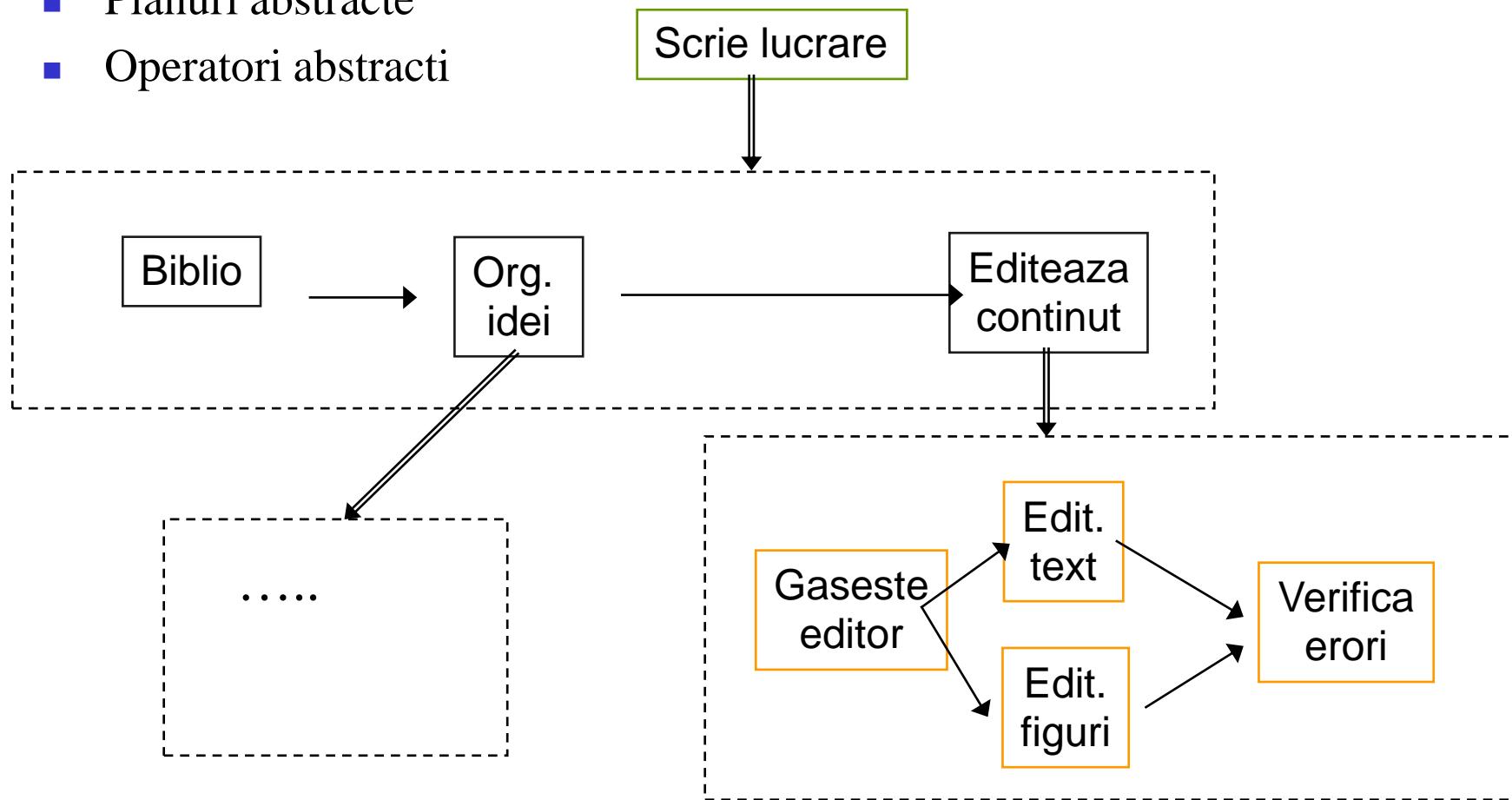
Criteriului adevarului necesar cu operatiile de modificare a planului



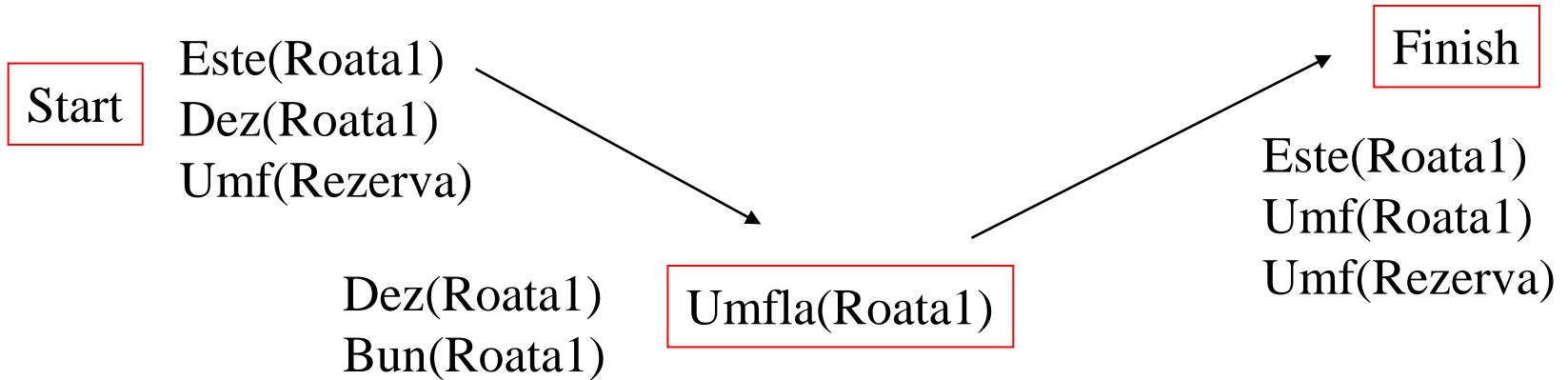
6. Planificare ierarhica

Planuri pe mai multe niveluri de abstractizare

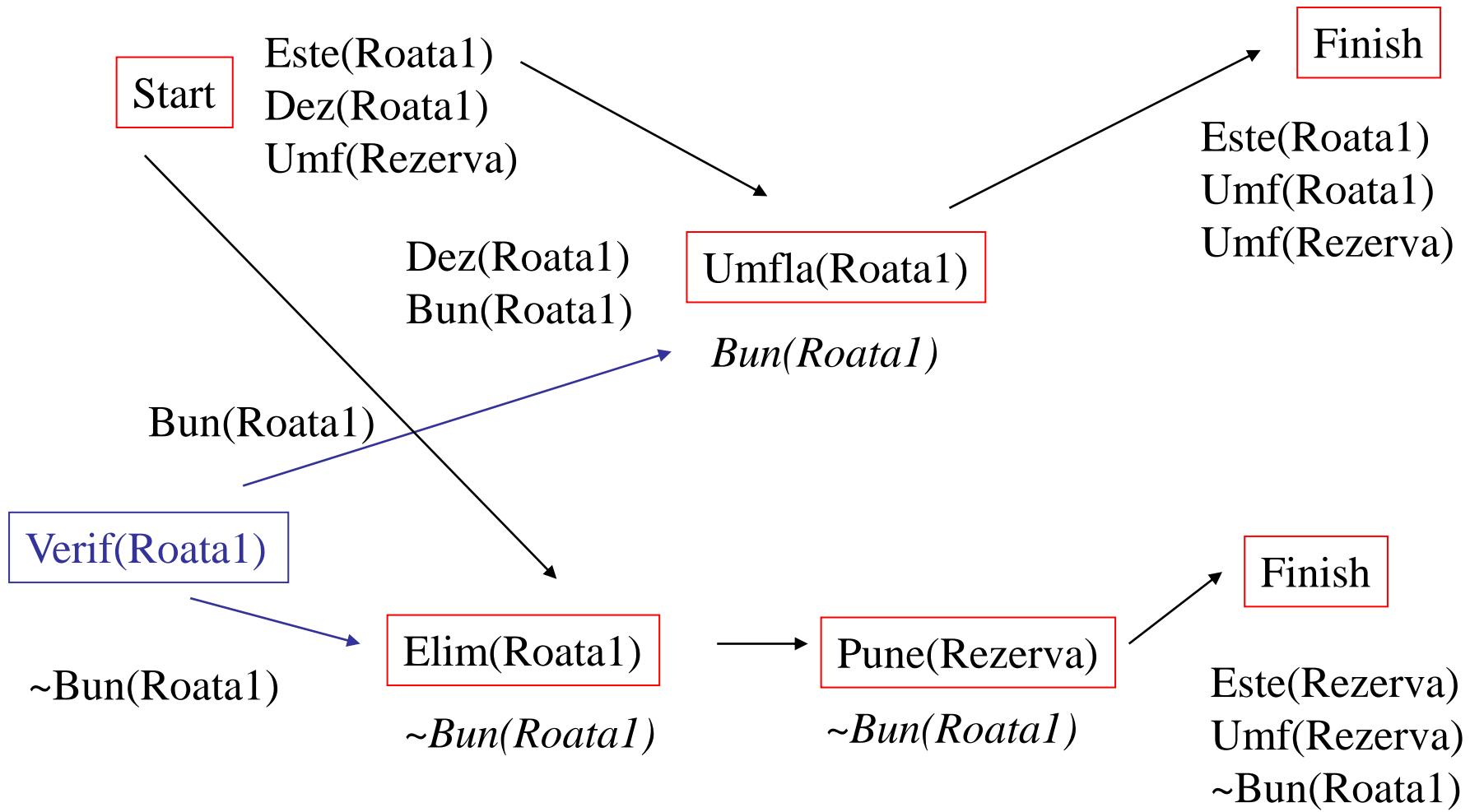
- Planuri abstracte
- Operatori abstracti



7. Planificare contingenta



Planificare contingenta - cont



Inteligentă Artificială

Universitatea Politehnica Bucuresti
Anul universitar 2020-2021

Adina Magda Florea



Curs nr. 8 si 9

Reprezentarea cunostintelor incerte

- Teoria probabilitatilor
- Retele Bayesiene
- Inferente exacte si aproximative in retele Bayesiene

1. Teoria probabilitatilor

1.1 Cunostinte incerte

$\forall p \text{ simpt}(p, \text{Dur_d}) \rightarrow \text{factor}(p, \text{carie})$

$\forall p \text{ simpt}(p, \text{Dur_d}) \rightarrow \text{factor}(p, \text{carie}) \vee \text{factor}(p, \text{infl_ging}) \vee \dots$

- LP
 - dificultate (« lene »)
 - ignoranta teoretica
 - ignoranta practica
- Teoria probabilitatilor \rightarrow un grad numeric de **incredere** sau **plauzibilitate** a afirmatiilor in $[0,1]$
- Gradul de adevar (fuzzy logic) \neq gradul de incredere

1.2 Definitii TP

- **Probabilitatea unui eveniment incert** A este masura gradului de incredere sau plauzibilitatea producerii unui eveniment
- Camp de probabilitate, S
- **Probabilitate neconditionata** (apriori) - inaintea obtinerii de probe pt o ipoteza / eveniment
- **Probabilitate conditionata** (aposteriori) - dupa obtinerea de probe

Exemple

$$P(\text{Carie}) = 0.1$$

$$P(\text{Vreme} = \text{Soare}) = 0.7$$

$$P(\text{Vreme} = \text{Ploaie}) = 0.2 \quad P(\text{Vreme} = \text{Nor}) = 0.1$$

Vreme - **variabila aleatoare**

- **Distributie de probabilitate**

Definitii TP - cont

- *Probabilitate conditionata* (aposteriori) - $P(A|B)$

$$P(\text{Carie} \mid \text{Dur_d}) = 0.8$$

Masura probabilitatii producerii unui eveniment A
este o functie $P:S \rightarrow R$ care satisface **axiomele**:

- $0 \leq P(A) \leq 1$
- $P(S) = 1$ (sau $P(\text{adev}) = 1$ si $P(\text{fals}) = 0$)
- $P(A \vee B) = P(A) + P(B) - P(A \wedge B)$

$$\begin{aligned} P(A \vee \sim A) &= P(A) + P(\sim A) - P(\text{fals}) = P(\text{adev}) \\ \Rightarrow P(\sim A) &= 1 - P(A) \end{aligned}$$

Definitii TP - cont

Evenimente **mutual exclusive**

Moneda – cap/pajura – mutual exclusive

Zar – 1, 2, 3 – mutual exclusive

Evenimente **exhaustive**

Moneda – cap/pajura – mutual exhaustive

Zar – 1, 2, 3, 4, 5, 6 – mutual exhaustive

Definitii TP - cont

A si B mutual exclusive \rightarrow

$$P(A \vee B) = P(A) + P(B)$$

$$P(e_1 \vee e_2 \vee e_3 \vee \dots e_n) =$$

$$P(e_1) + P(e_2) + P(e_3) + \dots + P(e_n)$$

$e(a)$ – multimea de evenimente atomice mutual exclusive si exhaustive in care apare a

$$P(a) = \sum_{e_i \in e(a)} P(e_i)$$

1.3 Regula produsului

Probabilitatea conditionata de producere a evenimentului A in conditiile producerii evenimentului B

- $P(A|B) = P(A \wedge B) / P(B)$

$$P(A \wedge B) = P(A|B) * P(B)$$

1.4 Teorema lui Bayes

$P(A|B) = P(A \wedge B) / P(B)$ – regula produsului

$$P(A|B) = P(A \wedge B) / P(B)$$

$$P(B|A) = P(A \wedge B) / P(A)$$



$$P(B|A) = P(A|B) * P(B) / P(A)$$

A_{Posteriori} = Plauzibilitate x A_{Priori} / Evidenta

Teorema lui Bayes

$$P(B|A) = P(A|B) * P(B) / P(A)$$

- Daca B si $\sim B$ sunt **mutual exclusive si exhaustive**, probabilitatea de producere a lui A in conditiile producerii lui B se poate scrie

$$P(A) = P(A \wedge B) + P(A \wedge \sim B) = P(A|B)*P(B) + P(A|\sim B)*P(\sim B)$$

$$\begin{aligned} P(B|A) &= \\ P(A | B) * P(B) / [P(A|B)*P(B) + P(A|\sim B)*P(\sim B)] \end{aligned}$$

Teorema lui Bayes

Generalizarea la mai multe ipoteze

B - h, A - e

$$P(h|e) = P(e | h) * P(h) / [P(e|h)*P(h) + P(e| \sim h)*P(\sim h)]$$

Daca h_i mutual exclusive si exhaustive, $i=1,k$

$$P(h_i|e) = \frac{P(e|h_i) \cdot P(h_i)}{\sum_{j=1}^k P(e|h_j) \cdot P(h_j)}, \quad i = 1, k$$

Teorema lui Bayes

Generalizarea la mai multe ipoteze si probe

h_i – evenimente / ipoteze probabile ($i=1,k$);

e_1, \dots, e_n – probe (evenimente)

$P(h_i)$

$P(h_i | e_1, \dots, e_n)$

$P(e_1, \dots, e_n | h_i)$

$$P(h_i | e_1, e_2, \dots, e_n) = \frac{P(e_1, e_2, \dots, e_n | h_i) \cdot P(h_i)}{\sum_{j=1}^k P(e_1, e_2, \dots, e_n | h_j) \cdot P(h_j)}, \quad i = 1, k$$

Teorema lui Bayes - cont

$$P(h_i|e_1, e_2, \dots, e_n) = \frac{P(e_1, e_2, \dots, e_n | h_i) \cdot P(h_i)}{\sum_{j=1}^k P(e_1, e_2, \dots, e_n | h_j) \cdot P(h_j)}, \quad i = 1, k$$

Daca e_1, \dots, e_n sunt ipoteze independente atunci

$$P(e|h_j) = P(e_1, e_2, \dots, e_n | h_j) = P(e_1 | h_j) \cdot P(e_2 | h_j) \cdots P(e_n | h_j), \quad j = 1, k$$

PROSPECTOR – sistem expert pentru consultari privind exploatari miniere si evaluarea resurselor

1.5 Inferente din DP si TB

Distributie de probabilitate

$\mathbf{P}(\text{Carie}, \text{Dur_d})$

| | Dur_d | \sim Dur_d |
|--------------|-------|--------------|
| Carie | 0.04 | 0.06 |
| \sim Carie | 0.01 | 0.89 |

$$P(\text{Carie}) = 0.04 + 0.06 = 0.1$$

$$P(\text{Carie} \vee \text{Dur}_d) = 0.04 + 0.01 + 0.06 = 0.11$$

$$P(\text{Carie} | \text{Dur}_d) = P(\text{Carie} \wedge \text{Dur}_d) / P(\text{Dur}_d) = 0.04 / 0.05$$

Inferente din DP si TB

| | Dur_d | | ~Dur_d | |
|--------|-------|-------|--------|-------|
| | Evid | ~Evid | Evid | ~Evid |
| Carie | 0.108 | 0.012 | 0.072 | 0.008 |
| ~Carie | 0.016 | 0.064 | 0.144 | 0.576 |

Distributie de probabilitate

P(Carie, Dur_d, Evid)

$$P(\text{Carie}) = 0.108 + 0.012 + 0.072 + 0.008 = 0.2$$

$$\begin{aligned} P(\text{Carie} \vee \text{Dur}_d) &= 0.108 + 0.012 + 0.072 + 0.008 + 0.016 \\ &+ 0.064 = 0.28 \end{aligned}$$

P(Carie, Dur_d, Vreme) – o tabela cu $2 \times 2 \times 3 = 12$ intrari
Distributie de probabilitate completa

Inferente din DP si TB

| | Dur_d | | ~Dur_d | |
|--------|-------|-------|--------|-------|
| | Evid | ~Evid | Evid | ~Evid |
| Carie | 0.108 | 0.012 | 0.072 | 0.008 |
| ~Carie | 0.016 | 0.064 | 0.144 | 0.576 |

$$P(\text{Carie} | \text{Dur_d}) = P(\text{Carie} \wedge \text{Dur_d}) / P(\text{Dur_d})$$

$$P(\sim\text{Carie} | \text{Dur_d}) = P(\sim\text{Carie} \wedge \text{Dur_d}) / P(\text{Dur_d})$$

$\alpha = 1 / P(\text{Dur_d})$ – constanta de normalizare a distributiei

$$P(\text{Carie} | \text{Dur_d}) = \alpha P(\text{Carie} \wedge \text{Dur_d}) =$$

$$\alpha [P(\text{Carie} \wedge \text{Dur_d} \wedge \text{Evid}) + P(\text{Carie} \wedge \text{Dur_d} \wedge \sim\text{Evid})] =$$

$$\alpha [<0.108, 0.016> + <0.012, 0.064>] = \alpha <0.12, 0.08> = <0.6, 0.4>$$

Chiar daca nu cunoastem α , adica $P(\text{Dur_d})$, putem calcula α

$$\alpha = 1/(0.12+0.08) = 1/0.2$$

Inferente din DP si TB

Generalizare – procedura generala de inferenta bazata pe DPC
Interogare asupra lui X

X – variabila de interogat (Carie)

E – lista de variabile probe (Dur_d)

e – lista valorilor observate pt aceste variabile E

Y – lista variabilelor neobservate (restul) (Evid)

$$P(\text{Carie} | \text{Dur_d}) = \alpha [P(\text{Carie} \wedge \text{Dur_d} \wedge \text{Evid}) + P(\text{Carie} \wedge \text{Dur_d} \wedge \neg \text{Evid})]$$

Insumarea se face peste toate combinatiile de valori ale variab. neobservate Y

$$P(X | e) = \alpha P(X, e) = \alpha \sum_{Y=y} P(X, e, Y)$$

Inferente din DP si TB

$$P(X | e) = \alpha P(X, e) = \alpha \sum_{Y=y} P(X, e, Y)$$

Avand DPC ecuatia poate da raspuns la interogari cu variabile discrete

Complex computational

n var Bool – tabela $O(2^n)$

- timp $O(2^n)$

1.6 Independenta conditională

- Evenimentele **X₁** și **X₂** sunt independente conditional fiind dat un eveniment **Y** daca

Stiind ca Y apare, aparitia lui X₁ nu influenteaza aparitia lui X₂ si aparitia lui X₂ nu influenteaza aparitia lui X₁

altfel spus

X₁ și **X₂** sunt independente conditional fiind dat un eveniment **Y** daca, pentru orice valoare a lui Y:

- distributia de probabilitate a lui X₁ este aceeasi pentru orice valoare ar lua X₂
- distributia de probabilitate a lui X₂ este aceeasi pentru orice valoare ar lua X₁

$$P(X_1, X_2 | Y) = P(X_1 | Y) * P(X_2 | Y)$$

Independenta conditională

$$P(\text{cauza} \mid \text{efect}) = P(\text{efect} \mid \text{cauza}) * P(\text{cauza}) / P(\text{efect})$$

$$P(y \mid x_1, \dots, x_n) = P(x_1, \dots, x_n \mid y) * P(y) / P(x_1, \dots, x_n)$$

$$P(y \mid x_1, \dots, x_n) = \alpha * P(x_1, \dots, x_n \mid y) * P(y)$$

Daca x_1, \dots, x_n sunt independente conditional fiind dat y atunci
 $P(x_1, \dots, x_n \mid y) = \prod_i P(x_i \mid y)$

$$\begin{aligned} P(\text{Cauza} \mid \text{Efect}_1, \text{Efect}_2, \dots) &= \\ \alpha P(\text{Cauza}) * \prod_i P(\text{Efect}_i \mid \text{Cauza}) \end{aligned}$$

1.7 Model Bayesian naiv

$$P(\text{Cauza}_y \mid \text{Efect}_1, \text{Efect}_2, \dots) = \alpha P(\text{Cauza}_y) * \prod_i P(\text{Efect}_i | \text{Cauza}_y)$$

Ipoteza de **independenta conditionala / naiva**

$$\text{Cauza_MAP} = \operatorname{argmax}_y P(\text{Cauza}_y) * \prod_i P(\text{Efect}_i | \text{Cauza}_y)$$

Model Bayesian naiv

Model Bayesian naiv

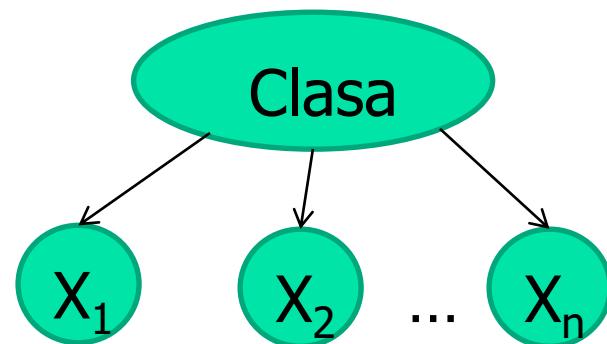
Vector de caracteristici x_1, \dots, x_n si o serie de clase C_k

$$P(C_k | x_1, x_2, \dots) = \alpha P(C_k) * \prod_i P(x_i | C_k)$$

$$C_{MAP} = \operatorname{argmax}_k P(C_k) * \prod_i P(x_i | C_k)$$

Model Bayesian naiv

MAP – Maximum a Posteriori



1.8 Modele grafice probabiliste

- Fiecare nod reprezinta o variabila aleatoare si fiecare legatura reprezinta o relatie probabilistica
 - Retele Bayesiene – modele grafice orientate - permit reprezentare compacta a DP si punerea in evidenta a independentei conditionale
 - Modele Markov – lanturi Markov – stari si tranzitii probabilistice

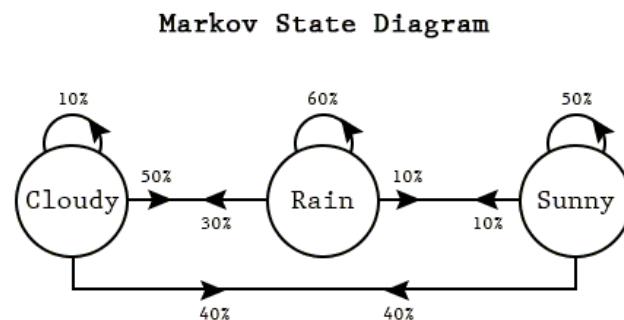
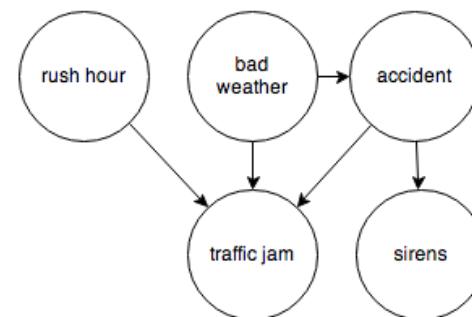


Figure 2



2 Retele Bayesiene

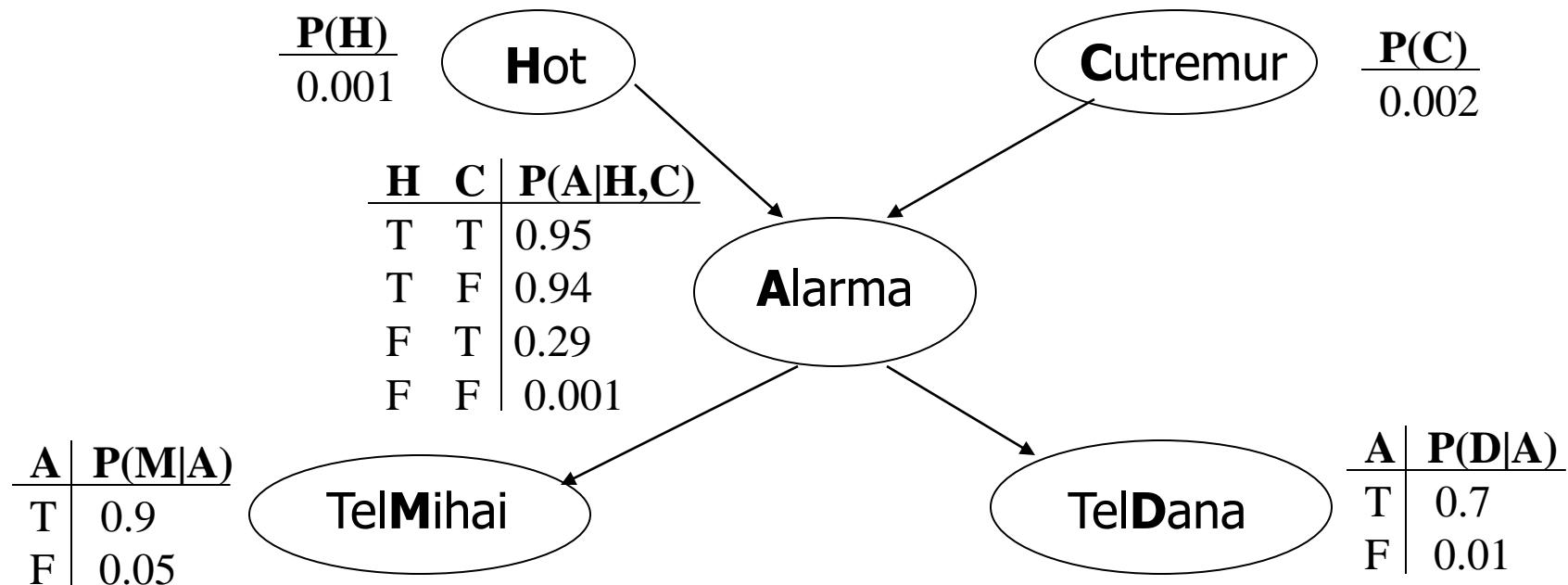
- Reprezinta dependente intre variabile aleatoare
- Specifica distributiei de probabilitate
- Simplifica calculele
- Au asociata o reprezentare grafica convenabila
- DAG care reprezinta relatiile cauzale intre variabile
- Pe baza structurii retelei se pot realiza diverse tipuri de inferente
- Calcule complexe in general dar se pot simplifica pentru structuri particulare

2.1 Structura retelelor Bayesiene

O RB este un DAG in care:

- Nodurile reprezinta variabilele aleatoare
- Legaturile orientate $X \rightarrow Y$: X are o influenta directa asupra lui Y, **X=Parinte(Y)**
- Fiecare nod are asociata o tabela de probabilitati conditionate care cuantifica efectul parintilor asupra nodului
 - $P(X_i | Parinti(X_i))$

Structura retelelor Bayesiene - cont



| | H | C | $P(A H, C)$ | |
|----------|---|---|---------------|-------|
| | | | T | F |
| TelMihai | T | T | 0.95 | 0.05 |
| TelMihai | T | F | 0.94 | 0.06 |
| TelMihai | F | T | 0.29 | 0.71 |
| TelMihai | F | F | 0.001 | 0.999 |

Tabela de probabilitati conditionate

Structura retelelor Bayesiene

In general X (Cauza) $\rightarrow Y$ (Efekt)

- Stabilesc topologia
- Specifica distributia de probabilitati conditionate
- Combinarea topologiei si distributia de probabilitati conditionate este suficienta pentru a specifica (implicit) intreaga DPC
- DPC poate raspunde la interogari
- Si RB la fel, mai eficient

2.2 Semantica retelelor Bayesiene

- Reprezentare a distributiei de probabilitate
- Specificare a independentei conditionale – constructia retelei
- Fiecare valoare din distributia de probabilitate poate fi calculata ca:

$$P(X_1=x_1 \wedge \dots \wedge X_n=x_n) = P(x_1, \dots, x_n) =$$

$$\prod_{i=1,n} P(x_i | \text{parinti}(x_i))$$

unde $\text{parinti}(x_i)$ reprezinta valorile specifice ale variabilelor $\text{Parinti}(X_i)$

2.3 Construirea retelei

Cum sa construim o retea a.i. RB/DPC sa fie o buna reprezentare?

Ecuatia $P(x_1, \dots, x_n) = \prod_{i=1,n} P(x_i | \text{parinti}(x_i))$ implica anumite relatii de independenta conditionala care pot ghida construirea retelei

$$P(X_1=x_1 \wedge \dots \wedge X_n=x_n) = P(x_1, \dots, x_n) =$$

$$P(x_n | x_{n-1}, \dots, x_1) * P(x_{n-1}, \dots, x_1) = \dots =$$

$$P(x_n | x_{n-1}, \dots, x_1) * P(x_{n-1} | x_{n-2}, \dots, x_1) * \dots * P(x_2 | x_1) * P(x_1) = \\ \prod_{i=1,n} P(x_i | x_{i-1}, \dots, x_1) - \text{valabila in general}$$

- DPC daca, pt fiecare variabila X_i din RB

$$P(X_i | X_{i-1}, \dots, X_1) = P(x_i | \text{Parinti}(X_i)) \text{ cu conditia ca} \\ \text{Parinti}(X_i) \subseteq \{ X_{i-1}, \dots, X_1 \}$$

Construirea retelei

Pt fiecare variabila X_i din RB

$P(X_i | X_{i-1}, \dots, X_1) = P(x_i | \text{Parinti}(X_i))$ cu
conditia ca

$$\text{Parinti}(X_i) \subseteq \{ X_{i-1}, \dots, X_1 \}$$

- O RB este o reprezentare corecta a domeniului cu conditia ca fiecare nod sa fie independent conditional de nondescendenti, fiind dati parintii lui

Construirea retelei

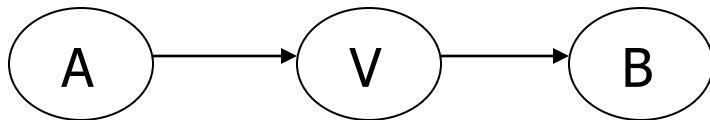
- Conditia poate fi satisfactuta prin etichetarea nodurilor intr-o ordine corespunzătoare DAG
- Intuitiv, parintii unui nod X_i trebuie sa fie toate acele noduri X_{i-1}, \dots, X_1 care influenteaza direct X_i .

Construirea retelei - cont

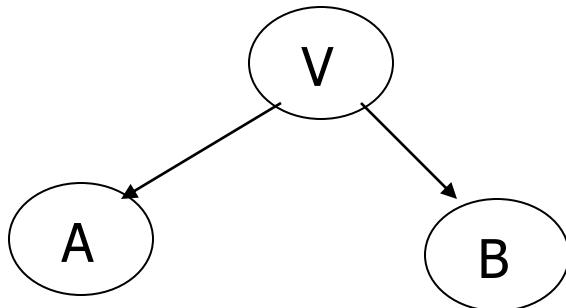
- Alege o multime de variabile aleatoare relevante care descriu problema
- Alege o ordonare a acestor variabile
- **cat timp** mai sunt variabile **repetă**
 - (a) alege o variabila X_i si adauga un nod corespunzator lui X_i
 - (b) atribuie $\text{Parinti}(X_i) \leftarrow$ un set minim de noduri deja existente in retea a.i. proprietatea de independenta conditionala este satisfacuta
 - (c) defineste tabela de probabilitati conditionate pentru X_i

Deoarece fiecare nod este legat numai la noduri anterioare → DAG

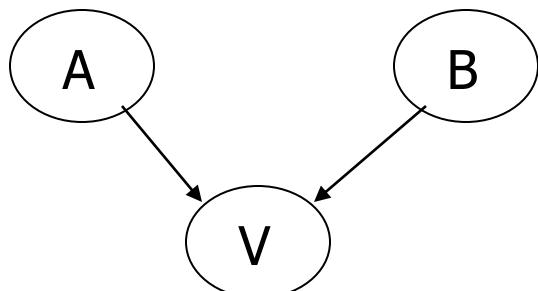
2.4 Inferente probabilistice



$$P(A \wedge V \wedge B) = P(A) * P(V|A) * P(B|V)$$

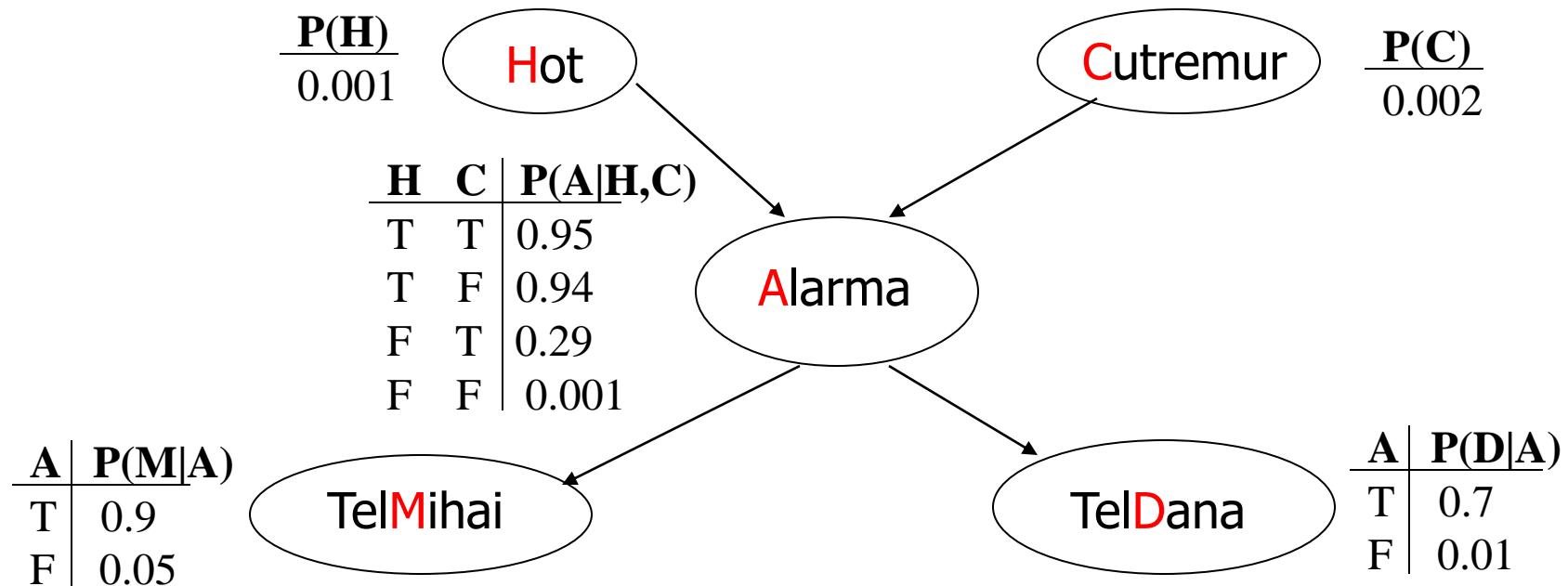


$$P(A \wedge V \wedge B) = P(V) * P(A|V) * P(B|V)$$



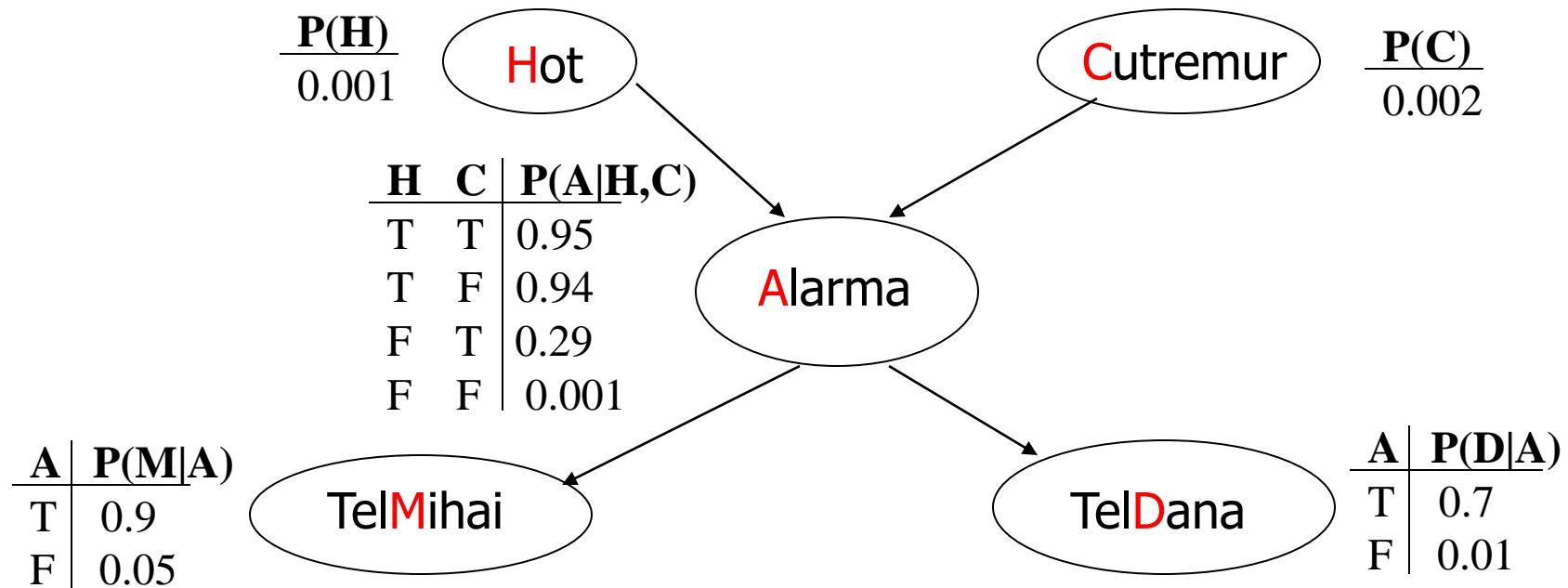
$$P(A \wedge V \wedge B) = P(A) * P(B) * P(V|A,B)$$

Inferente probabilistice



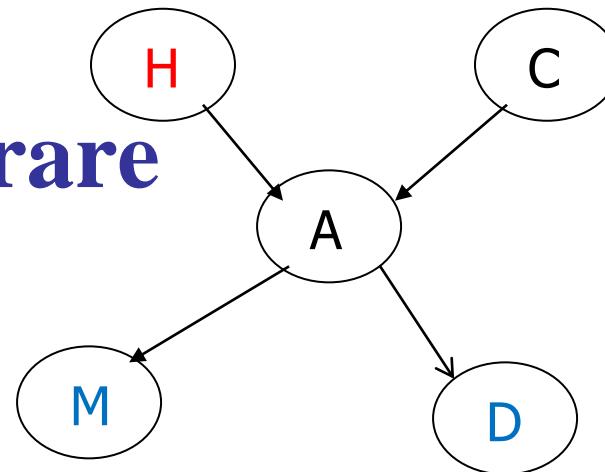
$$\begin{aligned}
 & P(M \wedge D \wedge A \wedge \neg H \wedge \neg C) = \\
 & P(M|A) * P(D|A) * P(A|\neg H \wedge \neg C) * P(\neg H) \wedge P(\neg C) = \\
 & 0.9 * 0.7 * 0.001 * 0.999 * 0.998 = 0.00062
 \end{aligned}$$

Inferențe probabilistice



$$\begin{aligned}
 P(A|H) &= P(A|H,C) * P(C|H) + P(A|H,\neg C) * P(\neg C|H) \\
 &= P(A|H,C) * P(C) + P(A|H,\neg C) * P(\neg C) \\
 &= 0.95 * 0.002 + 0.94 * 0.998 = 0.94002
 \end{aligned}$$

2.5 Inferență prin enumerare



X – variabila de interogare (Carie)

E – lista de variabile probe (Dur_d)

e – lista valorilor observate pt aceste variabile E

Y – lista variabilelor neobservate (restul) (Evid)

$$\mathbf{P}(X | e) = \alpha \mathbf{P}(X, e) = \alpha \sum_Y \mathbf{P}(X, e, Y)$$

$$\mathbf{P}(\mathbf{H|M,D}) = \alpha \mathbf{P}(\mathbf{H,M,D}) =$$

$$\alpha \sum_C \sum_A \mathbf{P}(\mathbf{H})\mathbf{P}(\mathbf{C})\mathbf{P}(\mathbf{A|H,C})\mathbf{P}(\mathbf{M|A})\mathbf{P}(\mathbf{D|A})$$

n var bool -> O(2ⁿ)

Inferență prin enumerare

$$P(H|M,D) = \alpha \sum_C \sum_A P(H)P(C)P(A|H,C)P(M|A)P(D|A)$$

$$\begin{aligned} P(H|M,D) &= \alpha P(H) \sum_C P(C) \sum_A P(A|H,C)P(M|A)P(D|A) \\ &= \alpha <0.00059224, 0.0014919> =_{\text{aprox}} <0.284, 0.716> \end{aligned}$$

Complexitate spatiu – $O(n)$

Complexitate timp $O(2^n)$

Inferență prin enumerare - algoritm

algoritm Enumerare(X,e,rb) intoarce distributie X

X – var de interogare

e – valori observate pt E

rb – Retea Bayesiana cu var $\{X\} \cup E \cup Y$

1. $Q(X) \leftarrow$ o distributie X, initial vida
 2. **pentru** fiecare valoare x_i a lui X **repetă**
 $Q(x_i) \leftarrow \text{EnumToate}(rb.Vars, e_{xi})$
unde e_{xi} este **e** extins cu $X=x_i$
 3. **intoarce** Normalizare($Q(x)$)
- sfarsit**

Inferență prin enumerare - algoritm

algoritm **EnumToate(Vars,e)** intoarce un număr real

1. daca $\text{Vars} = []$ atunci intoarce 1.0

2. $Y \leftarrow \text{first}(\text{Vars})$

3. **daca** Y are valoare y în e

atunci intoarce $P(y|\text{parinti}(Y)) * \text{EnumToate}(\text{Rest}(\text{Vars}), e)$

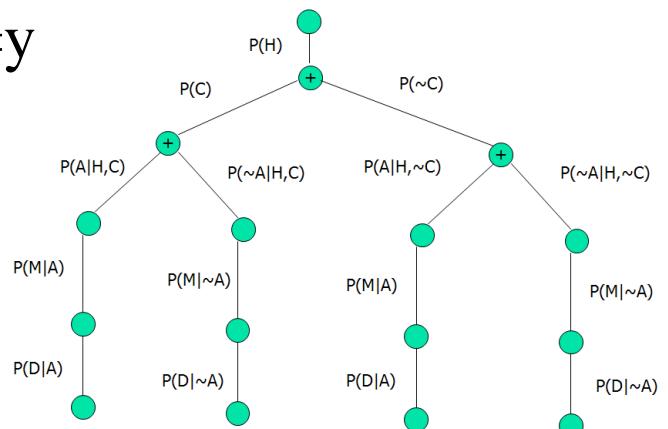
altfel intoarce

$\sum_y P(y|\text{parinti}(Y)) * \text{EnumToate}(\text{Rest}(\text{Vars}), e_y)$

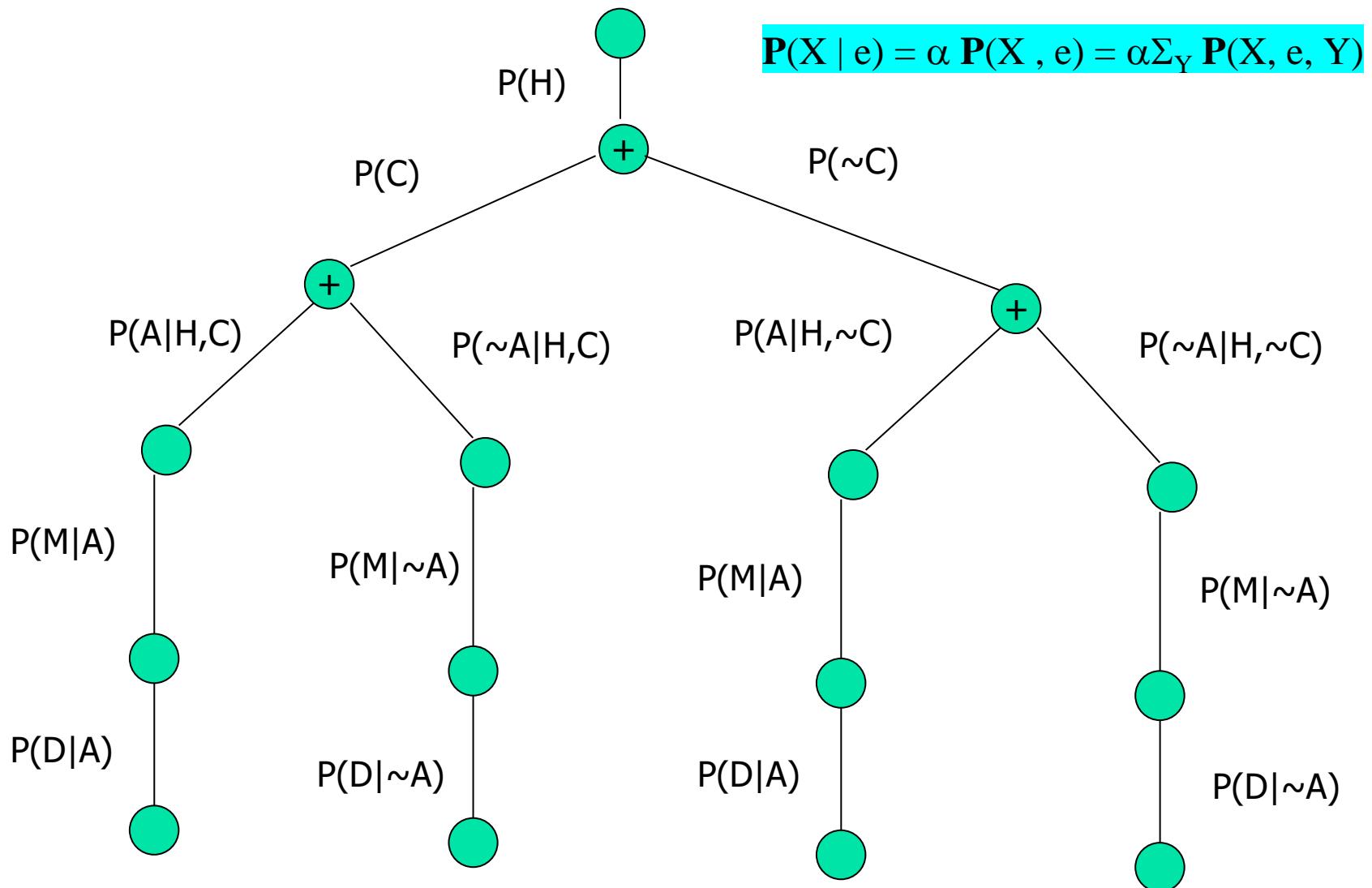
unde e_y este e extins cu $Y=y$

sfarsit

$$P(\mathbf{H|M,D}) = \alpha \sum_C \sum_A P(H)P(C)P(A|H,C)P(M|A)P(D|A)$$



$P(M|A)P(D|A)$ si $P(M|\sim A)P(D|\sim A)$ computed 2 times



$$\mathbf{P}(\mathbf{H}|\mathbf{M},\mathbf{D})= \alpha \sum_C \sum_A P(H)P(C)P(A|H,C)P(M|A)P(D|A)$$

Eliminarea variabilelor (Variable elimination)

Pt a elimina calculele duplicate – se calculeaza o data si se salveaza rezultatul

Idee – se calculeaza ecuatia de la dreapta la stanga

$$P(H|M,D) = \alpha \sum_C \sum_A P(H) P(C) P(A|H,C) P(M|A) P(D|A)$$

$f_1(H) \quad f_2(C) \quad f_3(A,H,C) \quad f_4(A) \quad f_5(A)$

$$f_5(A) = [P(D|A) \quad P(D|\sim A)] = [.07 \quad .001]$$

$$f_4(A) = [P(M|A) \quad P(M|\sim A)] = [.09 \quad .05]$$

$f_3(A,H,C)$ va fi o matrice de $2 \times 2 \times 2$

$$[P(A|H,C) \quad P(\sim A|H,C) \quad P(A|H, \sim C) \quad P(\sim A|H, \sim C) \dots P(\sim A|\sim H, \sim C)]$$

Eliminarea variabilelor

$$P(H|M,D) = \alpha \sum_C \sum_A P(H) P(C) \frac{P(A|H,C) P(M|A) P(D|A)}{f_1(H) f_2(C) f_3(A,H,C) f_4(A) f_5(A)}$$

Insumam peste A in f3, f4, f5 – rezulta un factor $f_6(H,C)$

$$f_6(H,C) = \sum_A f_3(A,H,C) \times f_4(A) \times f_5(A) =$$

$$f_3(a,H,C) \times f_4(a) \times f_5(a) + f_3(\sim a,H,C) \times f_4(\sim a) \times f_5(\sim a)$$

Insumam peste C – rezulta un factor $f_7(H)$

$$P(H|M,D) = \alpha f_1(H) \times \underline{\sum_C f_2(C) \times f_6(H,C)}$$

$$f_7(H) = \sum_C f_2(C) \times f_6(H,C) = f_2(c) \times f_6(H,c) + f_2(\sim c) \times f_6(H,\sim c)$$

$$P(H|M,D) = \alpha f_1(H) \times f_7(H)$$

Avem nevoie de 2 operatii

- Pointwise product \times
- Insumarea unei variabile dintr-un produs de factori

Implementarea operațiilor

Pointwise product

$$(\mathbf{f}_1 \times \mathbf{f}_2)(X_1, \dots, X_i, Y_1, \dots, Y_j, Z_1, \dots, Z_k) =$$

$$\mathbf{f}_1(X_1, \dots, X_i, Y_1, \dots, Y_j) \times \mathbf{f}_2(Y_1, \dots, Y_j, Z_1, \dots, Z_k)$$

Insumarea unei variabile dintr-un produs de factori

- Insumarea peste X_1 , rezultatul este un factor peste X_2, \dots, X_i

$$(\Sigma_{X_1} \mathbf{f})(X_2, \dots, X_i) = \mathbf{f}(X_1=v_1, \dots, X_i) + \dots + \mathbf{f}(X_1=v_k, \dots, X_i)$$

Implementarea operațiilor - exemplu

Pointwise product

$f1(A,B) \times f2(B,C) = f3(A,B,C)$ are $2^{1+1+1} = 8$ intrari

| A | B | f1(A,B) | B | C | f2(B,C) | A | B | C | f3(A,B,C) |
|---|---|---------|---|---|---------|---|---|---|---------------|
| a | a | .3 | a | a | .2 | a | a | a | .3 x .2 = .06 |
| a | f | .7 | a | f | .8 | a | a | f | .3 x .8 = .24 |
| f | a | .9 | f | a | .6 | a | f | a | .7 x .6 = .42 |
| f | f | .1 | f | f | .4 | a | f | f | .7 x .4 = .28 |
| | | | | | | f | a | a | .9 x .2 = .18 |
| | | | | | | f | a | f | .9 x .8 = .72 |
| | | | | | | f | f | a | .1 x .6 = .06 |
| | | | | | | f | f | f | .1 x .4 = .04 |

Implementarea operațiilor - exemplu

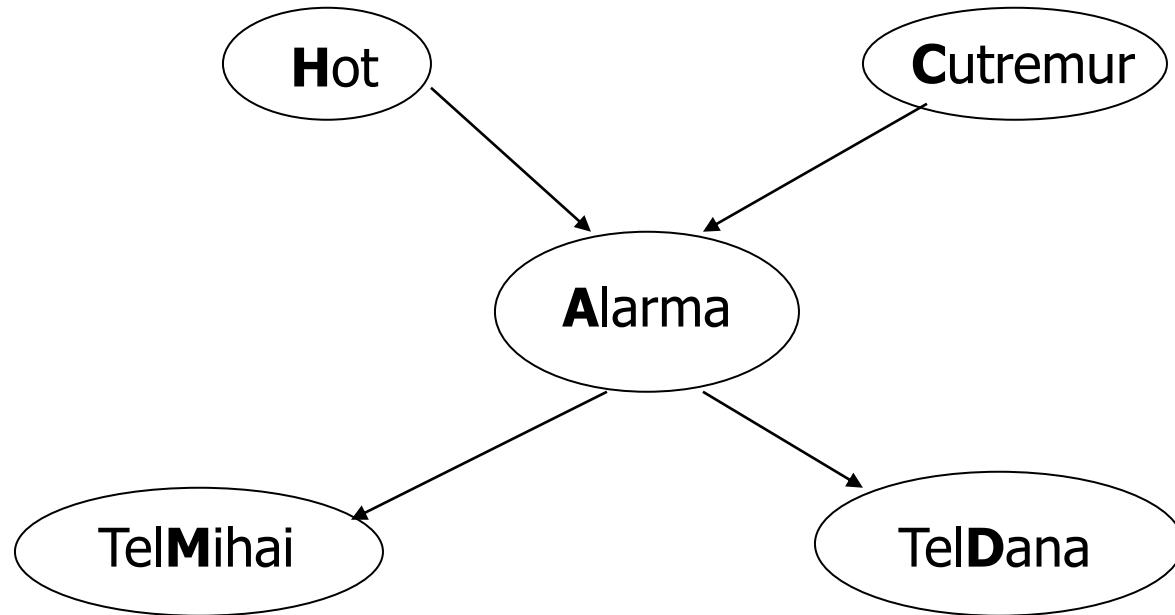
Insumarea unei variabile dintr-un produs de factori

- Insumarea peste A din $f3(A,B,C)$

$$f(B,C) = \sum_A f3(A,B,C) = f3(a,B,C) + f3(\sim a,B,C) =$$

$$\begin{vmatrix} .06 & .24 \\ .42 & .28 \end{vmatrix} + \begin{vmatrix} .18 & .72 \\ .06 & .04 \end{vmatrix} = \begin{vmatrix} .24 & .96 \\ .48 & .32 \end{vmatrix}$$

2.6 Tipuri de inferență



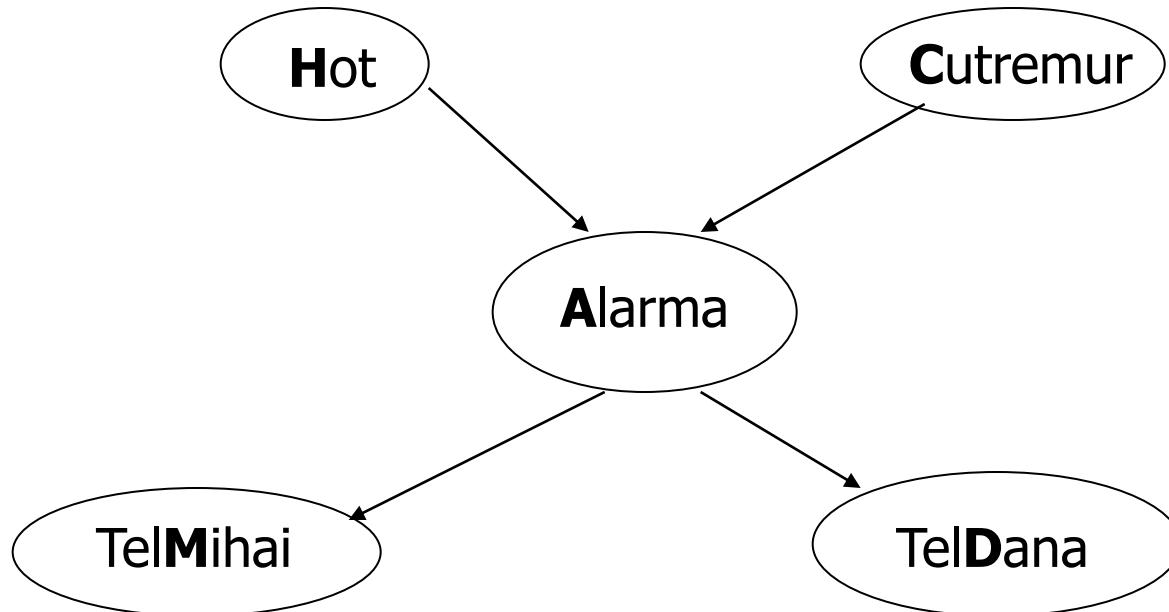
Inferente de diagnosticare (efect → cauza)

$$P(\text{Hot} | \text{TelMihai})$$

Inferente cauzale (cauza → efect)

$$P(\text{TelMihai} | \text{Hot}), P(\text{TelDana} | \text{Hot})$$

Tipuri de inferență



Inferente intercauzale (intre cauza si efecte comune)

$$P(\text{Hot} | \text{Alarma} \wedge \text{TelDana})$$

Inferente mixte

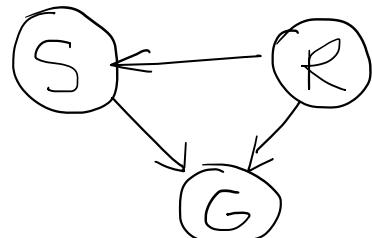
$$P(\text{Alarma} | \text{TelMihai} \wedge \neg \text{Cutremur}) \rightarrow \text{diag + cauzal}$$

$$P(\text{Hot} | \text{TelMihai} \wedge \neg \text{Cutremur}) \rightarrow \text{diag + intercauzal}$$

2.7 Independenta in RB

- Evenimentele **X si Y sunt independente conditional fiind dat un eveniment Z daca**
Stiind ca Z apare, aparitia lui X nu influenteaza aparitia lui Y si aparitia lui Y nu influenteaza aparitia lui X
- Structura de graf RB codifica anumite relatii de independenta: fiecare nod este independent conditional de nondescendenti, fiind dati parintii lui

$$P(A, B | C) = P(A|C) \cdot P(B|C)$$



$G_{\text{indep cond de } R}$ si nodul S

$$P(x_1 \dots x_n) = \prod P(x_i | \text{Par}_k(x_i))$$



Ef causa

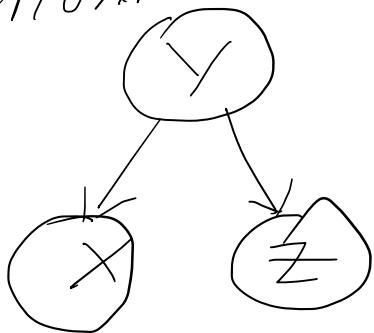
$Y \text{ observe}, Z \text{ mind } X$

$Y \text{ parent } X \Rightarrow$

$X \text{ insulation} \Leftrightarrow \text{not find } Z \text{ and } Y$

CAUZA

COMUNA



$(X \perp Z | Y)$

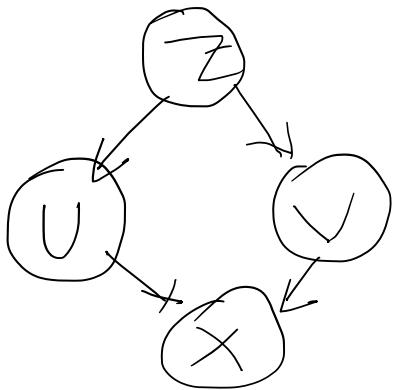
Stim Y ($Y \text{ observe}$)

$Z \text{ random } X$

$Y \text{ par } \text{mix } X$

$\Rightarrow X \text{ si } Z \text{ sent independent}$

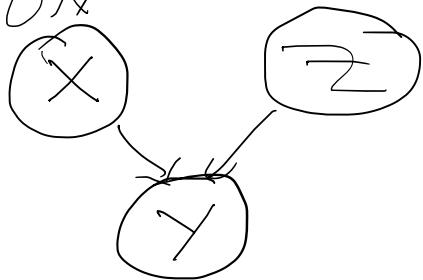
$\text{fus } \text{not } Y$
 $(X \perp Z | Y)$



$(X \perp Z | U)$? NU
 $(X \perp Z | U, V)$? DA

EFFECT

COCAINE



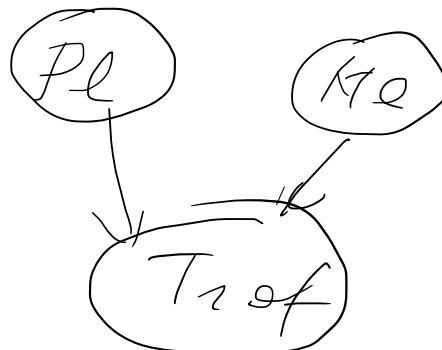
X mu are for

Z consider all X

Nucleophile Z
 $(X \perp Z | Y)$

Daother will see Z

$\cancel{(X \perp Z | Y)}$ NU



Independenta in RB

- Mai sunt si alte relatii de independenta?
- Doua sau mai multe relatii de independenta conditionala pot conduce la o noua relatie folosind un mecanism bazat pe axiome grafoide
- **D-separabilitatea** este un concept care surprinde astfel de relatii de independenta conditionala derivate
- Z **d-separa** pe X si Y intr-un DAG daca X si Y sunt independente conditional fiind dat Z dpv al axiomelor grafoide.
- **(X ⊥ Y | Z) – X este independent conditional de Y fiind dat Z**

D-separabilitate

D-separabilitate

- Cand nu este indeplinita $(X \perp Y | Z)$?

Conexiuni directe $X \rightarrow Y$

- Se pot influenta indiferent de Z

Conexiuni indirecte: X si Y nu sunt direct conectate dar exista un lant intre ele (trail)

D-separabilitate

- Fie cazul a 3 noduri

Caz a: Efect cauzal

Lantul cauzal $X \rightarrow Z \rightarrow Y$

- X nu influenteaza Y via Z daca Z observat



Caz b: Efect evidenta

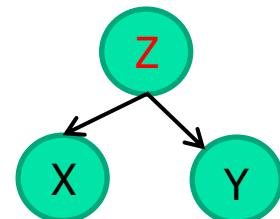
- $Y \rightarrow Z \rightarrow X$
- Identic cu cazul a:
- X nu influenteaza Y via Z daca Z observat
- **daca $(X \perp Y | Z)$ nu este indeplinit atunci nici $(Y \perp X | Z)$ nu este indeplinit**



D-separabilitate

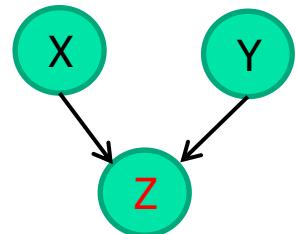
Caz c: Cauza comună

- $X \leftarrow Z \rightarrow Y$
- X nu influenteaza pe Y via Z daca Z observat



Case d: Efect comun

- $X \rightarrow Z \leftarrow Y$
- Influenta nu poate merge de-a lungul lantului $X \rightarrow Z \leftarrow Y$ daca Z nu este observat

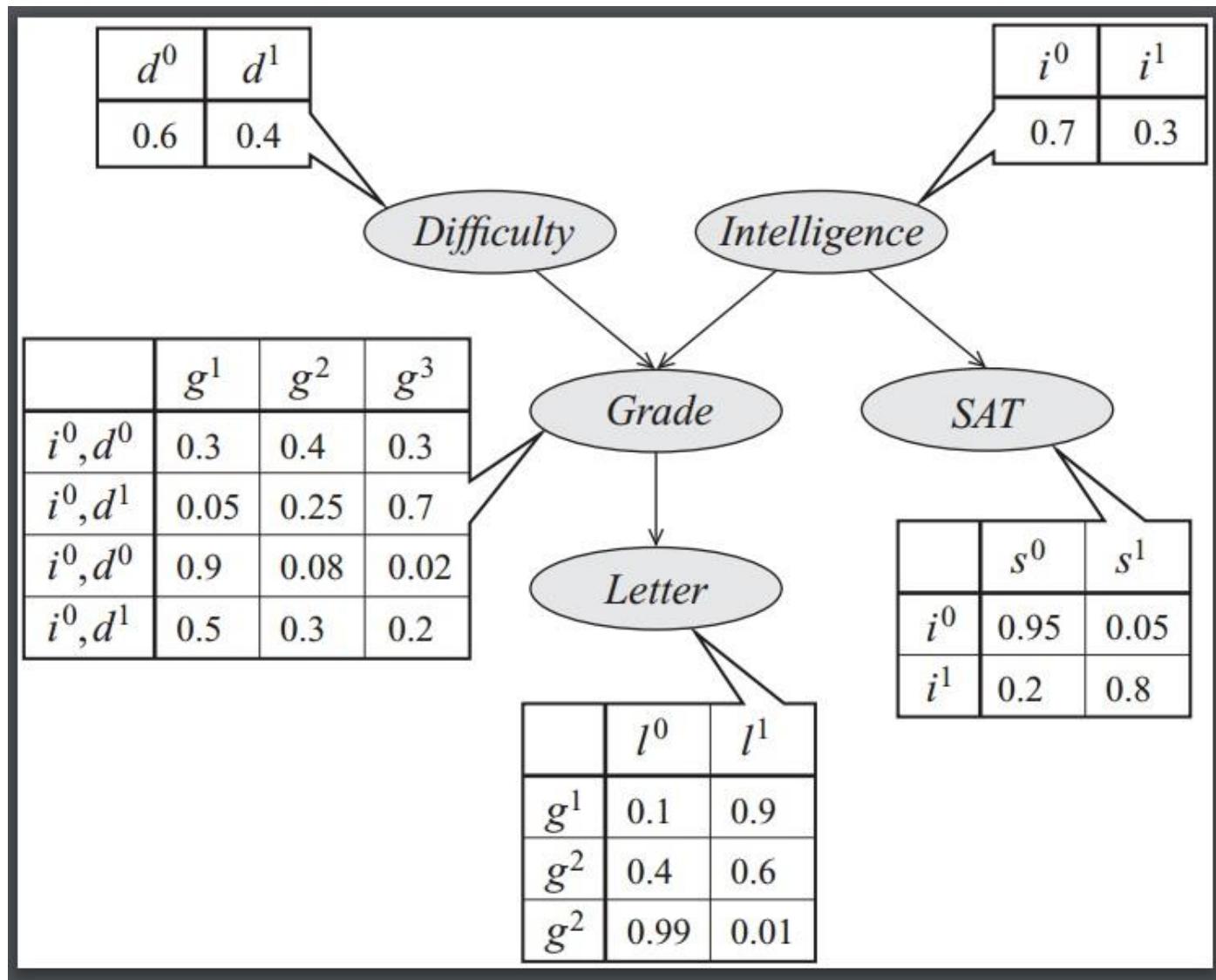


Deci X nu influenteaza pe Y via Z daca Z NU este observat

D-separabilitate

Daca influenta poate curge de la X la Y via Z spunem ca lantul $X \leftrightarrow Z \leftrightarrow Y$ este activ

- Efect cauzal $X \rightarrow Z \rightarrow Y$ – activ $\leftrightarrow Z$ nu este observat
- Efect evidenta $Y \rightarrow Z \rightarrow X$ – activ $\leftrightarrow Z$ nu este observat
- Cauza comună $X \leftarrow Z \rightarrow Y$ – activ $\leftrightarrow Z$ nu este observat
- Efect comun $X \rightarrow Z \leftarrow Y$ - activ \leftrightarrow fie Z fie unul din descendentei lui Z este observat
- **Influenta probabilistica = flux in graf** – cand influenta din X poate curge prin Z catre Y a.i. sa afecteze probabilitatea lui Y ,
- Deci X nu este conditional independent de Y fiind dat Z



2.8 Inferențe de aproximare în RB

- Inferente exacte în RB:
 - Inferenta prin enumerare
 - Eliminarea variabilelor (eficientizare)

Asa cum s-a aratat anterior, exprimam probabilitatea conditionata ca probabilitate neconditionata, apoi insumam peste variabilele neobserve

$$P(X | e) = \alpha P(X, e) = \alpha \sum_Y P(X, e, Y)$$

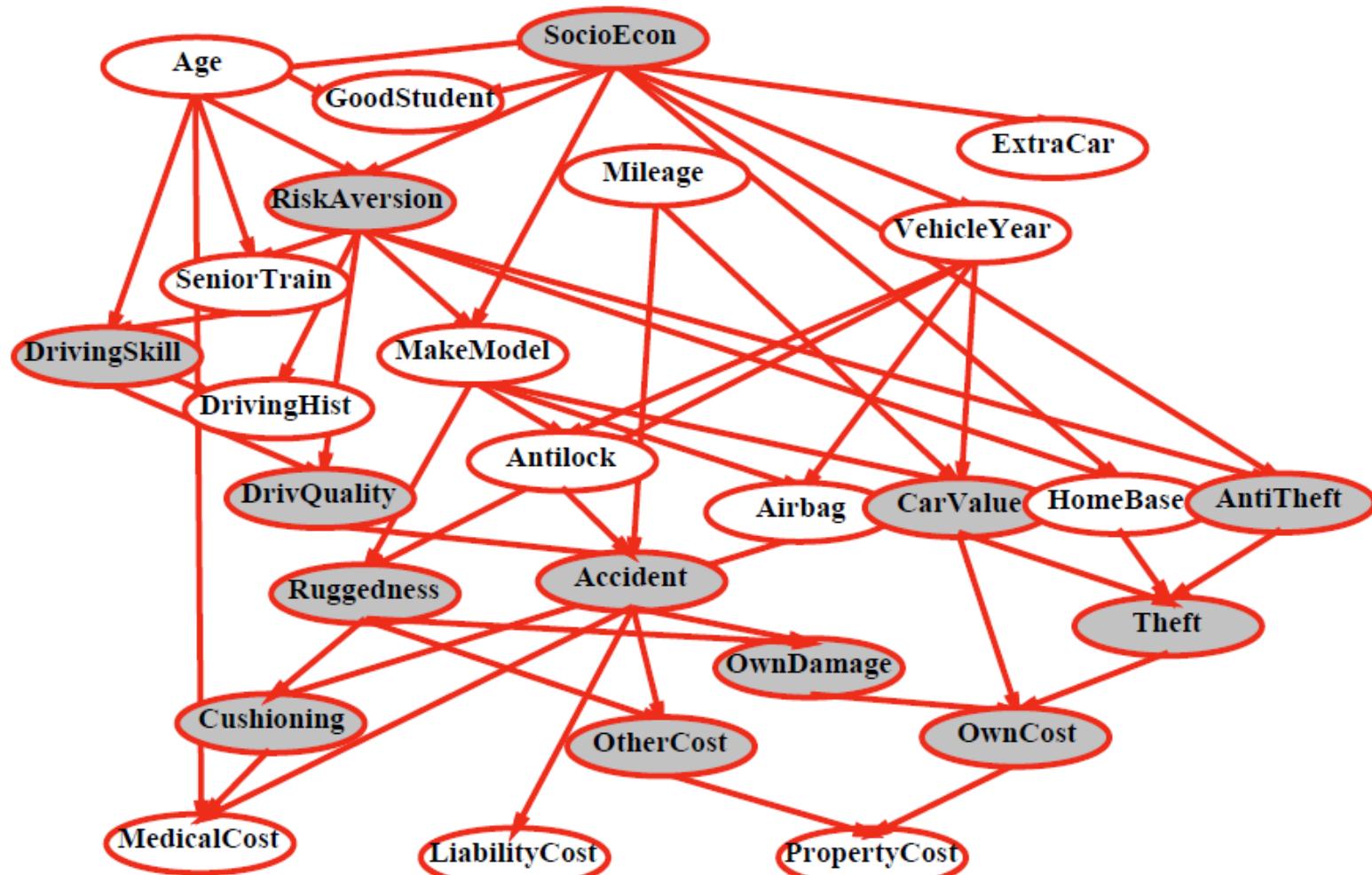
Inferente de aproximare in RB

Inferente exacte in RB foarte costisitoare

- Complexitate spatiu – $O(n)$
- Complexitate timp $O(2^n)$ – pt n variabile booleene

Pentru retele complexe aceste metode devin de multe ori nepractice

Asigurarea auto



Inferențe de aproximare

Inferente de aproximare

- Algoritmi de esantionare aleatoare, cunoscuti si ca algoritmi Monte Carlo
- Ofera raspunsuri aproximative care depind de numarul de esantioane care se genereaza
- Ne intereseaza esantionare pentru calculul probabilitatilor conditionate

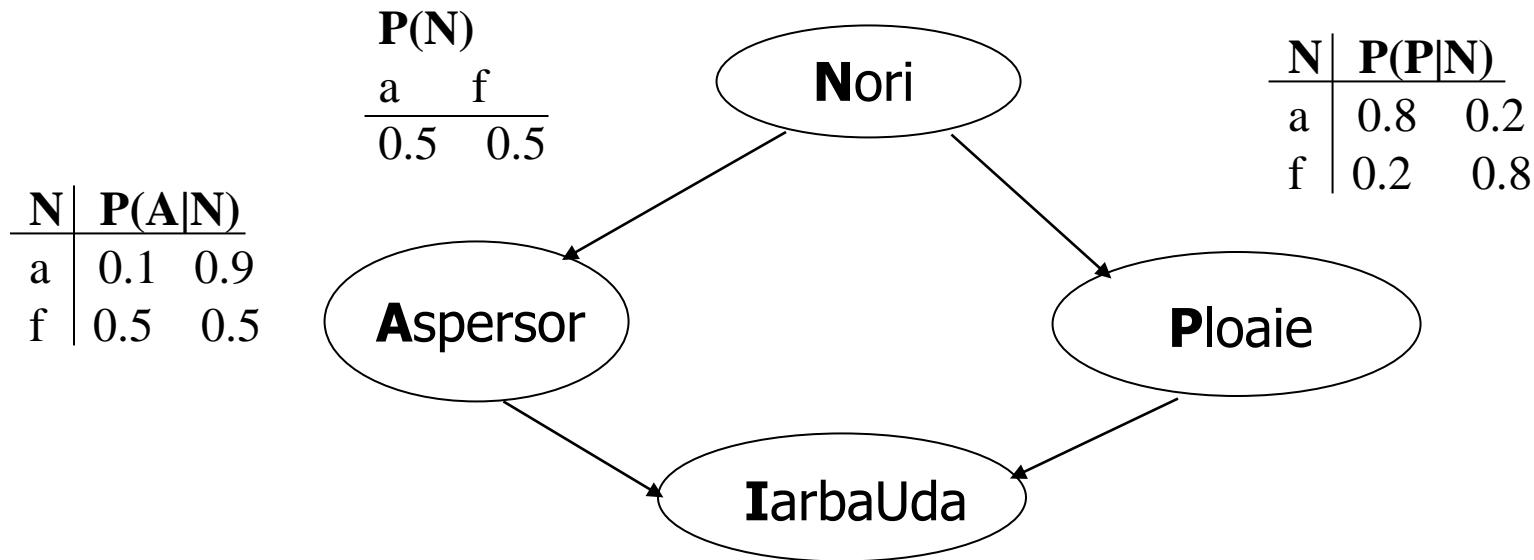
Esantionare directă

Esantionare cu lanturi Markov

a) Esantionare directă

- Se genereaza esantioane dintr-o distributie de probabilitate cunoscuta
- Cel mai simplu proces de esantionare intr-o RB genereaza esantioane/evenimente din retea fara nici o proba
- Ideea este de a esantiona fiecare variabila pe rand in ordine topologica
- Distributia de probabilitate din care se obtin valorile unei variabile din esantion este conditionata de valorile care au fost deja atribuite parintilor

RB exemplu



| A | P | P(I A, P) | |
|---|---|-------------|------|
| | | a | f |
| a | a | 0.99 | 0.01 |
| a | f | 0.9 | 0.1 |
| f | a | 0.9 | 0.1 |
| f | f | 0.1 | 0.9 |

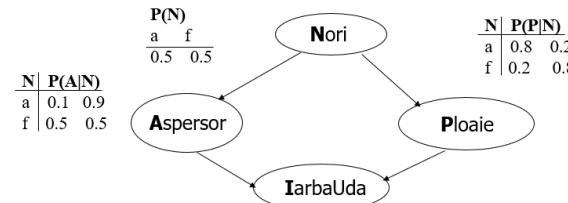
Generarea esantionului

Presupun ordinea $[Nori, Aspersor, Ploaie, IarbaUda]$

1. Esantion $\mathbf{P(Nori)} = \langle 0.5 \ 0.5 \rangle$ a
2. Esantion din $\mathbf{P(Aspersor|Nori=a)} = \langle 0.1 \ 0.9 \rangle$ f
3. Esantion $\mathbf{P(Ploaie|Nori=a)} = \langle 0.8 \ 0.2 \rangle$ a
4. Esantion din $\mathbf{P(IarbaUda|Aspersor=f, Ploaie=a)} = \langle 0.9 \ 0.1 \rangle$ a

Esantionarea este **[a, f, a, a]**

RB exemplu



| A | P | P(I A, P) | |
|---|---|-------------|------|
| a | a | 0.99 | 0.01 |
| a | f | 0.9 | 0.1 |
| f | a | 0.9 | 0.1 |
| f | f | 0.1 | 0.9 |

Algoritm de esantionare

**algoritm Esantionare(rb) intoarce esantion
intrari:** rb specificand d.p. $P(X_1, \dots, X_N)$

$x \leftarrow$ esantion cu n elemente

pentru fiecare variabila X_i din X_1, \dots, X_N **repeta**

$x[i] \leftarrow$ un esantion aleator din $P(X_i | \text{parinti}(X_i))$

intoarce x

sfarsit

Algoritm de esantionare

- Intr-un algoritm de esantionare, raspunsurile sunt calculate prin numararea numarului de esantioane generate
- Sa presupunem ca avem N esantionari

$N_{PS}(x_1, \dots, x_n)$ – de cate ori a aparut x_1, \dots, x_n in multimea de esantioane

- Se presupune ca acest numar va converge chiar la probabilitate
$$\lim_{N \rightarrow \infty} (N_{PS}(x_1, \dots, x_n)/N) = P(x_1, \dots, x_n)$$
- Fie S_{PS} probabilitatea ca un eveniment(esantion) specific sa fie generat de algoritmul de esantionare

$$S_{PS}(x_1, \dots, x_n) = \prod_{i=1,n} P(x_i | \text{parinti}(X_i)) = P(x_1, \dots, x_n)$$

deoarece fiecare pas de esantionare depinde de valorile parintilor

Algoritm de esantionare

$$S_{PS}(x_1, \dots, x_n) = \prod_{i=1,n} P(x_i | \text{parinti}(X_i)) = P(x_1, \dots, x_n)$$

$$\lim_{N \rightarrow \infty} (N_{PS}(x_1, \dots, x_n)/N) = S_{PS}(x_1, \dots, x_n) = P(x_1, \dots, x_n)$$

De exemplu sa consideram esantionul

[a, f, a, a] generat anterior

Probabilitatea de esantionare a acestui eveniment

$$S_{PS}(a,f,a,a) = 0.5 \times 0.9 \times 0.8 \times 0.9 = 0.324$$

Pentru N foarte mare, putem presupune ca 32,4% din esantioane vor fi egale cu acesta

b) Eliminarea esantioanelor nepotrivite

Eliminarea esantioanelor (*Rejection sampling*)

- Se refera la situatia in care avem de estimat o probabilitate conditionata $P(X|e)$
- Se genereaza esantioanele din distributia de probabilitate specificata de retea
- Apoi se elimina acele esantioane care nu se potrivesc cu probele e
- In final estimarea lui $P(X=x|e)$ este obtinuta prin numararea numarului de aparitii $X=x$ in esantioanele ramase

Eliminarea esantioanelor nepotrivite

- Dorim sa estimam

$$P(\text{Ploaie}|\text{Aspersor} = a)$$

- Generam, de ex, 100 esantioane
- Presupunem ca din cele 100 esantioane 73 au Aspersor = f, le eliminam
- Raman 27 cu Aspersor = a
- Din acestea 8 au Ploaie=a si 19 au Ploaie=f
- $P(\text{Ploaie}|\text{Aspersor} = a) = \text{Normalizare}(<8, 19>) = <0.296, 0.704>$

Raspunsul corect este <0.3,0.7>

- Pe masura ce colectam mai multe esantioane, raspunsul va converge catre valoarea reala

Algoritm de esantionare cu eliminare

algoritm EsantionareEliminare(X,e,rb,N)

intoarce o estimare $P(X|e)$

intrari: rb specificand d.p. $P(X_1, \dots, X_N)$

X – variabila de interogare

e – valorile observate pentru variabilele E

N – numarul total de esantioane generate

variabile locale VN – vector care numara aparitiile valorilor lui X

pentru $j \leftarrow 1, N$ repeta

$y \leftarrow$ Esantionare (rb)

daca y este consistent cu e **atunci**

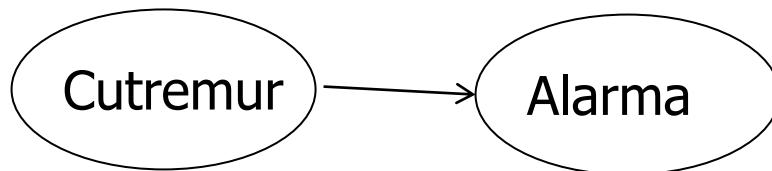
$VN[x] \leftarrow VN[x]+1$ unde x este valoarea lui X in y

intoarce Normalizare(VN)

sfarsit

Eliminarea esantioanelor nepotrivite

- Principala problema a acestei abordari este faptul ca elimina prea multe esantioane



- Esantioane pt $P(\text{Cutremur}|\text{Alarma}=a)$

~~Cutremur =f, Alarma =f~~

~~Cutremur =f, Alarma =f~~

~~Cutremur =f, Alarma =f~~

Cutremur =a, Alarma = a

- Procentul de esantioane consistent cu probele scade exponential pe masura ce creste numarul de probe, deci problematic pentru probleme complexe

c) Esantionarea ponderata

Esantionare ponderata (*Likelihood weighting*)

- Elimina ineficienta potentiala a eliminarii esantioanelor prin generarea numai a acelor **evenimente** (esantioane) care sunt **consistente cu probele e**
- Se fixeaza valorile variabilelor probe E si se esantioneaza numai pentru celelalte variabile
- Acest lucru garanteaza faptul ca fiecare esantion generat este consistent cu probele; dar nu toate evenimentele sunt la fel de importante

Esantionarea ponderata

- Înainte de a le numara, fiecare eveniment este ponderat cu o **plauzibilitate** (likelihood) pe care esantionul o acorda probei
- **Plauzibilitatea (likelihood)** este masurată ca **produsul probabilităților conditionate pentru fiecare variabilă probă, fiind date parintii ei**

Esantionarea ponderata - Exemplu

$$P(\text{Ploaie} | \text{Nori}=a, \text{IarbaUda}=a)$$

si ordonarea $[\text{Nori}, \text{Aspersor}, \text{Ploaie}, \text{IarbaUda}]$

Initial $w=1.0$

Se genereaza un esantion

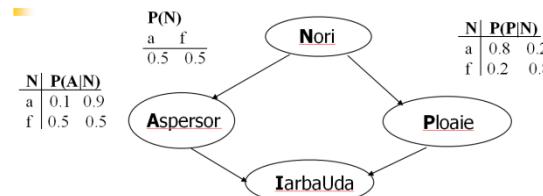
1. *Nori* este o variabila proba cu valoarea **a**

In consecinta $w \leftarrow w \times P(\text{Nori}=a) = 0.5$

2. *Aspersor* nu este o variabila proba deci esantionam

$$P(\text{Aspersor} | \text{Nori}=a) = (0.1, 0.9) - \text{presupun } f$$

3. Similar $P(\text{Ploaie} | \text{Nori}=a) = (0.8, 0.2) - \text{presupun } a$



| A | P | $P(I A, P)$ | |
|---|---|---------------|------|
| a | a | 0.99 | 0.01 |
| a | f | 0.9 | 0.1 |
| f | a | 0.9 | 0.1 |
| f | f | 0.1 | 0.9 |

Esantionarea ponderata - Exemplu

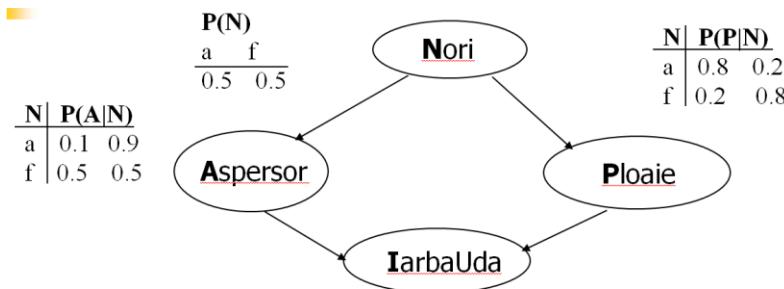
$$P(\text{Ploaie} | \text{Nori}=a, \text{IarbaUda}=a)$$

si ordonarea $[\text{Nori}, \text{Aspersor}, \text{Ploaie}, \text{IarbaUda}]$

4. IarbaUda este o variabila proba cu valoarea **a**

$$\mathbf{w} \leftarrow \mathbf{w} \times P(\text{IarbaUda}=a | \text{Aspersor}=f, \text{Ploaie}=a) = \\ \mathbf{0.9} \times 0.5 = 0.45$$

Esantionarea ponderata intoarce un esantion **[a,f,a,a]** cu **ponderea 0.45** in conditiile $\text{Ploaie}=a$



| A | P | $P(I A, P)$ | |
|---|---|---------------|------|
| | | a | f |
| a | a | 0.99 | 0.01 |
| a | f | 0.9 | 0.1 |
| f | a | 0.9 | 0.1 |
| f | f | 0.1 | 0.9 |

Algoritm de esantionare ponderata

algoritm EsantionarePonderata(X,e,rb,N)

intoarce o estimare $P(X|e)$

intrari: rb specificand d.p. $P(X_1, \dots, X_N)$

X – variabila de interogare

e – valorile observate pentru variabilele E

N – numarul total de esantioane generate

variabile locale W – vector care numara aparitiile valorilor lui X, ponderate

pentru $j \leftarrow 1, N$ **repeta**

$x, w \leftarrow$ Esantion-Ponderat(rb,e)

$W[x] \leftarrow W[x] + w$ unde x este valoarea lui X in x

intoarce Normalizare(W)

sfarsit

Algoritm de esantionare ponderata

algoritm Esantion-Ponderat(rb,e)

intoarce un esantion si o pondere

$w \leftarrow 1$

$x \leftarrow$ un esantion cu n elemente initializate din e

pentru fiecare variabila X_i din X_1, \dots, X_n **repetă**

daca X_i este o variabila proba cu valoarea x_i in e

atunci $w \leftarrow w * P(X_i=x_i | \text{parinti}(X_i))$

altfel $x[i] \leftarrow$ un esantion aleator din $P(X_i | \text{parinti}(X_i))$

intoarce x, w

sfarsit



Inteligentă Artificială

Universitatea Politehnica Bucuresti
Anul universitar 2020-2021

Adina Magda Florea



Curs nr. 10

Agenti inteligenți

1. Definitii ale agentilor in stiinta calculatoarelor

- Nu exista o definitie unanim acceptata
- IA, agenti intelectuali, sisteme multi-agent
- Aparent agentii sunt dotati cu intelect
- Sunt toti agentii intelectuali?
- **Agent** = definit mai mult prin caracteristici, unele pot fi considerate ca manifestari ale unui comportament intelectual

Definitii agenti

- “De cele mai multe ori, oamenii folosesc termenul agent pentru a referi o entitate care functioneaza **permanent** si **autonom** intr-un mediu in care exsita alte procese si/sau alti agenti”
- “Un agent este o entitate care **percepe** mediul in care se afla si **actioneaza** asupra acestuia”

- “**Agent** = un sistem (software sau hardware) cu urmatoarele proprietati:
 - **autonomie** – agentii opereaza fara interventi directa a utilizatorui si au un anumit control asupra actiunilor si starilor lor;
 - **reactivitate**: agentii percep mediul si reacioneaza corespunzator al schimbarile din acesta;
 - **pro-activitate**: agentii, pe langa reactia la schimbarile din mediu, sunt capabili sa urmareasca executia scopurilor si sa actioneze independent;
 - **abilitati sociale** – agentii interactioneaza cu altri agenti sau cu utilizatorul pe baza unui limbaj de comunicare.

2. Caracteristici agenti

2 directii de definitie

- Definirea unui agent izolat
- Definirea agentilor in colectivitate →
dimensiune sociala → SMA

2 tipuri de definitii

- Nu neaparat agenti intelectuali
- Include o comportare tipica IA → **agenti intelectuali**

Caracteristici agenti

- Actioneaza pentru un utilizator sau un program
- Autonomie
- Percepe mediul si actioneaza asupra lui reactiv
- Actiuni pro-active
- Caracter social
- Functionare continua (persistent software)
- Mobilitate

inteligenta?

- **Scopuri, rationalitate**
- **Rationament, luarea deciziilor** *cognitiv*
- **Invatare/adaptare**
- **Interactiune cu alți agenti – dimensiune socială**

Alte moduri de a realiza inteligenta?

SMA – mai multi agenti in acelasi mediu

- **Interactiuni intre agenti**
 - nivel inalt
- Interactiuni pentru- coordonare
 - comunicare
 - organizare

□ **Coordonare**

- ➔ motivati colectiv
- ➔ motivati individual
- scopuri proprii / indiferenta
- scopuri proprii / competitie pentru resurse
- scopuri proprii si contradictorii / competitie pentru resurse
- scopuri proprii / coalitii

□ Comunicare

- protocol
- limbaj

- negociere
- ontologii

□ Structuri organizationale

- centralize vs decentralize
 - ierarhie/ piata
- abordare *"agent cognitiv"*

3 Tipuri de agenti

3.1 Agenti cognitivi

Modelul uman al perspectivei asupra lumii → caracterizare agent utilizand reprezentari simbolice și *notiuni mentale*

- knowledge - cunoștințe
- beliefs - convingeri
- desires, goals – dorințe, scopuri
- intentions - intentii
- commitments - angajamente
- obligations - obligații

3.2 Agenti reactivi

- Unitati simple de prelucrare care percep mediul si reacioneaza la schimbarile din mediu
- Nu folosesc reprezentari simbolice sau rationament.
- Inteligenta nu este situata la nivel individual ci distribuita in sistem, rezulta din interactiunea entitatilor cu mediu – “emergence”

3.3 Exemple de probleme tipice

Problema inteleptilor

Regele picteaza cate o pata alba si spune ca cel putin o pata este alba



Dilema prizonierului

Rezultatele pentru A si B (in puncte ipotetice) in functie de combinatiile de actiuni execute

| Player A / Player B | Defect | Cooperate |
|--------------------------------|---------------|------------------|
| Defect | 2 , 2 | 5 , 0 |
| Cooperate | 0 , 5 | 3 , 3 |

Problema prazilor si vanatorilor



Abordare cognitiva

- Vanatorii au scopuri, prazole nu
- Detectia prazilor
- Echipa vanatori, roluri
- Comunicare/cooperare

Abordare reactiva

- Pazole emit semnale a caror intensitate scade pe masura cresterii distantei de vanatori
- Vanatorii emit semnale care pot fi percepute de alti vanatori
- Fiecare vanator este atras de o prada si respins de alt semnal de la un vanator

- **Cautare distribuita**
- **Planificare distribuita**

4. Modele arhitecturale de agenti

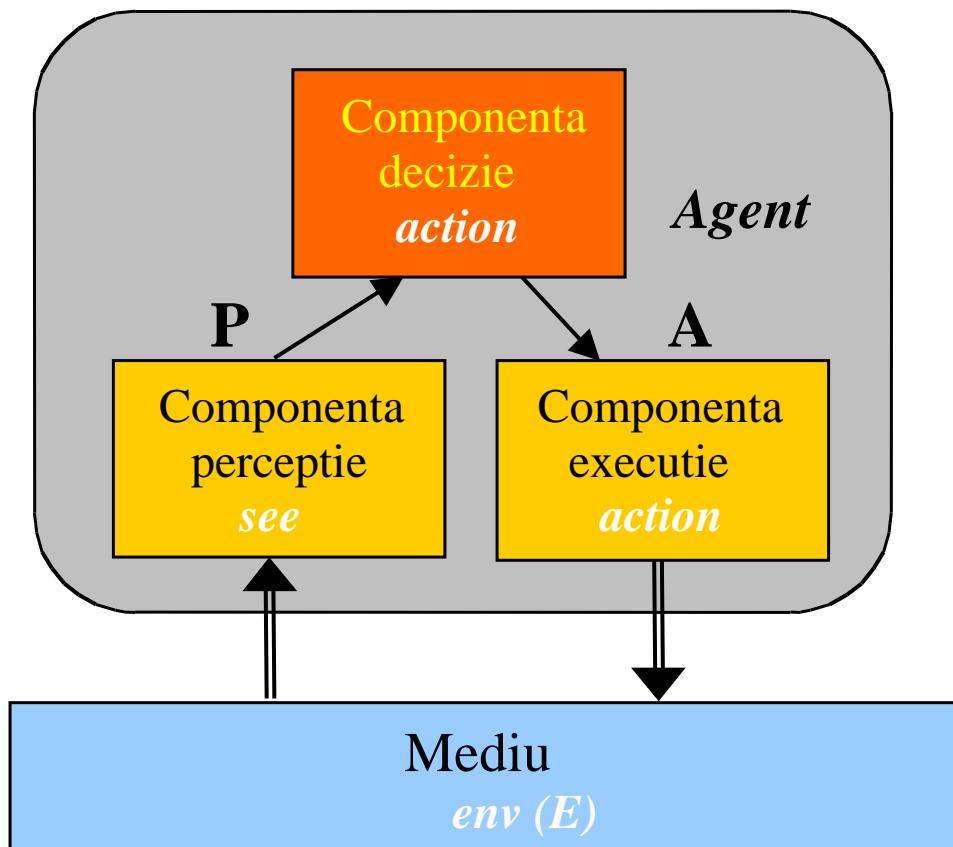
- Structura conceptuala a agentilor
- Arhitecturi de agenti cognitivi
- Arhitecturi de agenti reactivi

4.1 Structura conceptuala a agentilor

Rationalitatea unui agent

- Ce inseamna rationalitatea unui agent
- *Cum putem masura rationalitatea unui agent?*
- O masura a performantei
- Arhitectura simpla -> se detaliaza treptat

Modelare agent reactiv

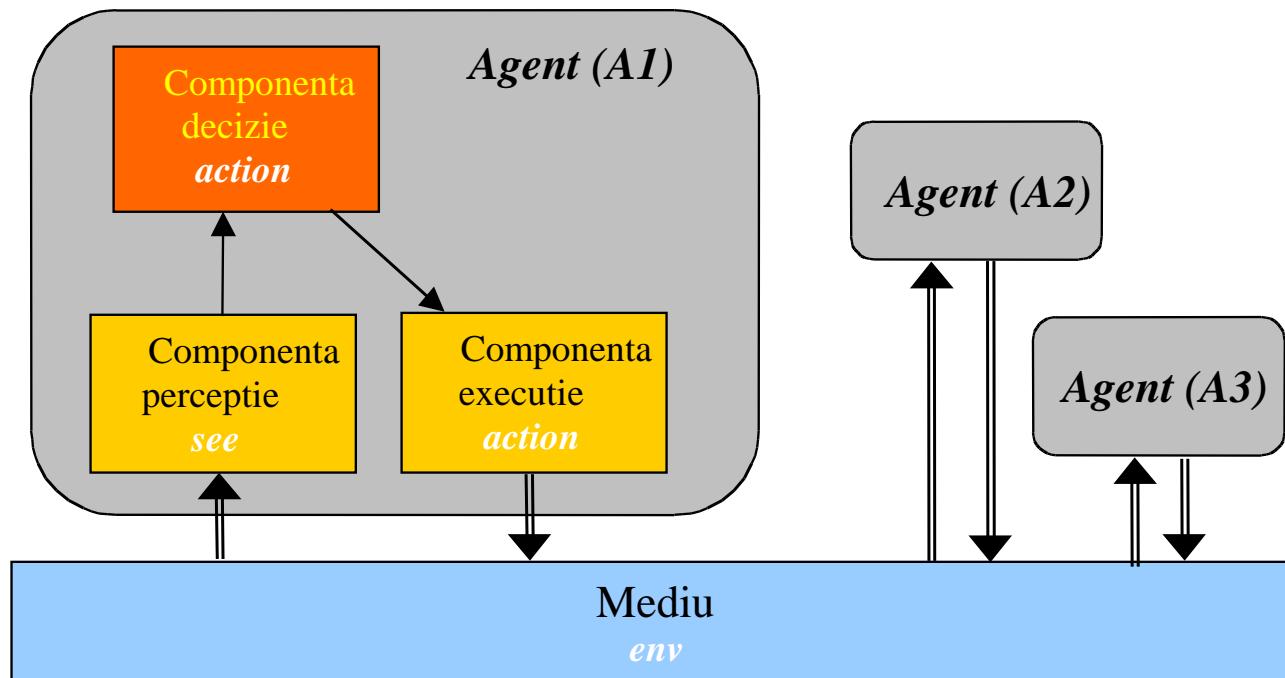


$$\begin{aligned} E &= \{e_1, \dots, e, \dots\} \\ P &= \{p_1, \dots, p, \dots\} \\ A &= \{a_1, \dots, a, \dots\} \end{aligned}$$

Agent reactiv
 $\text{see} : E \rightarrow P$
 $\text{action} : P \rightarrow A$
 $\text{env} : E \times A \rightarrow E$
 $(\text{env} : E \times A \rightarrow P(E))$

Modelare agenti reactivi

A_1, \dots, A_i, \dots
 P_1, \dots, P_i, \dots
(de obicei identice)



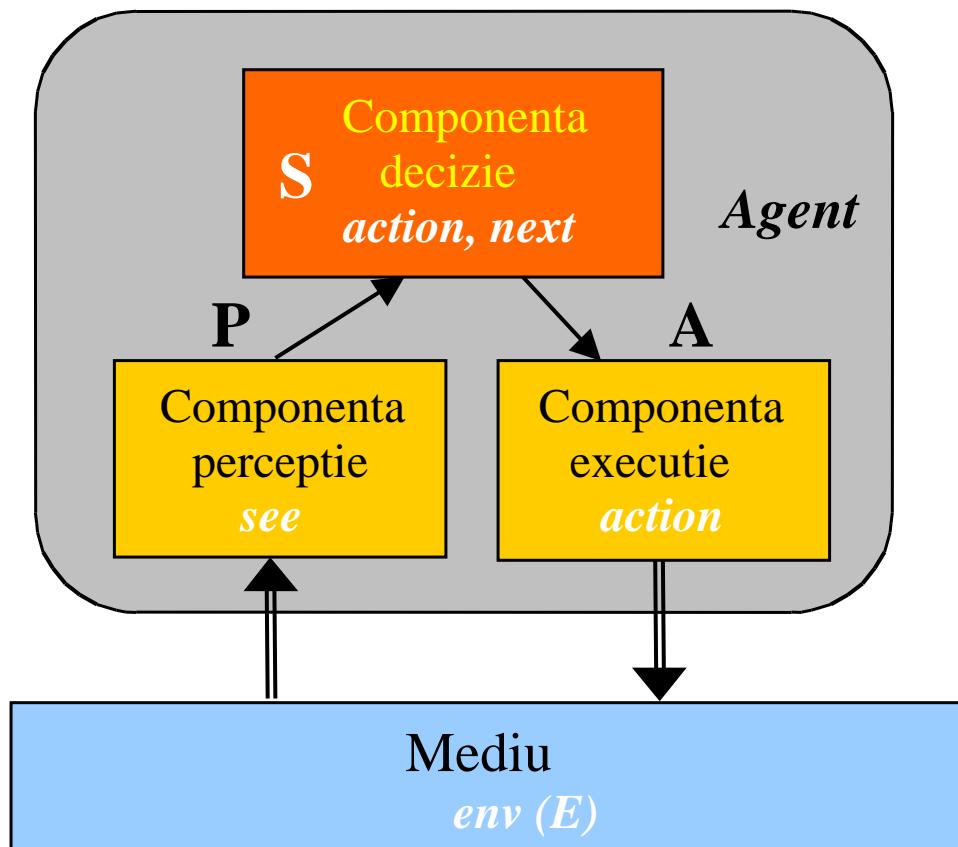
Mai multi agenti reactivi

$$see_i : E \rightarrow P_i$$

$$action_i : P_i \rightarrow A_i$$

$$env : E \times A_1 \times \dots \times A_n \rightarrow P(E)$$

Modelare agent cognitiv



$$\begin{aligned} E &= \{e_1, \dots, e, \dots\} \\ P &= \{p_1, \dots, p, \dots\} \\ A &= \{a_1, \dots, a, \dots\} \\ S &= \{s_1, \dots, s, \dots\} \end{aligned}$$

Agent cu stare
 $see : E \rightarrow P$
 $next : S \times P \rightarrow S$
 $action : S \rightarrow A$
 $env : E \times A \rightarrow P(E)$

Modelare agenti cognitivi

S_1, \dots, S_i, \dots

A_1, \dots, A_i, \dots

P_1, \dots, P_i, \dots

(nu intotdeauna identice)

$I = \{i_1, \dots, i_k, \dots\}$

Mai multi agenti cognitivi

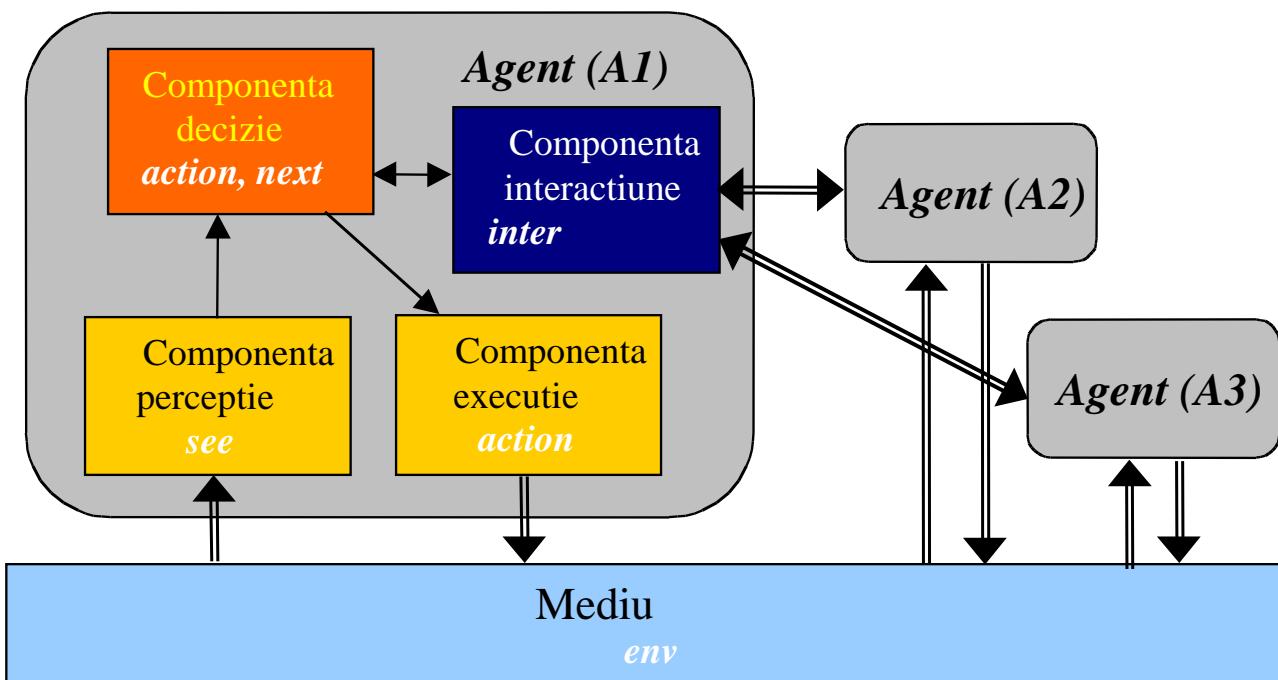
$see_i : E \rightarrow P_i$

$next_i : S_i \times P \rightarrow S_i$

$action_i : S_i \times I \rightarrow A_i$

$inter_i : S_i \rightarrow I$

$env : E \times A_1 \times \dots \times A_n \rightarrow P(E)$



Modelare agent cognitiv

Agenti cu stare si scopuri

$goal : E \rightarrow \{0, 1\}$

Agenti cu utilitate

$utility : E \rightarrow R$

Mediu nedeterminist

$env : E \times A \rightarrow P(E)$

Probabilitatea estimata de un agent ca rezultatul unei actiuni (a) executata in e sa fie noua stare e'

$$\sum_{e' \in env(e, a)} prob(ex(a, e') = e') = 1$$

Modelare agent cognitiv

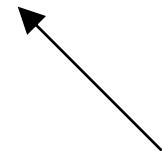
Agenti cu utilitate

Utilitatea estimata (*expected utility*) a unei actiuni a intr-o stare e , dpv al agentului

$$U(a, e) = \sum_{e' \in env(e, a)} prob(ex(a, e) = e') * utility(e')$$

Principiul utilitatii estimate maxime

Maximum Expected Utility (MEU)



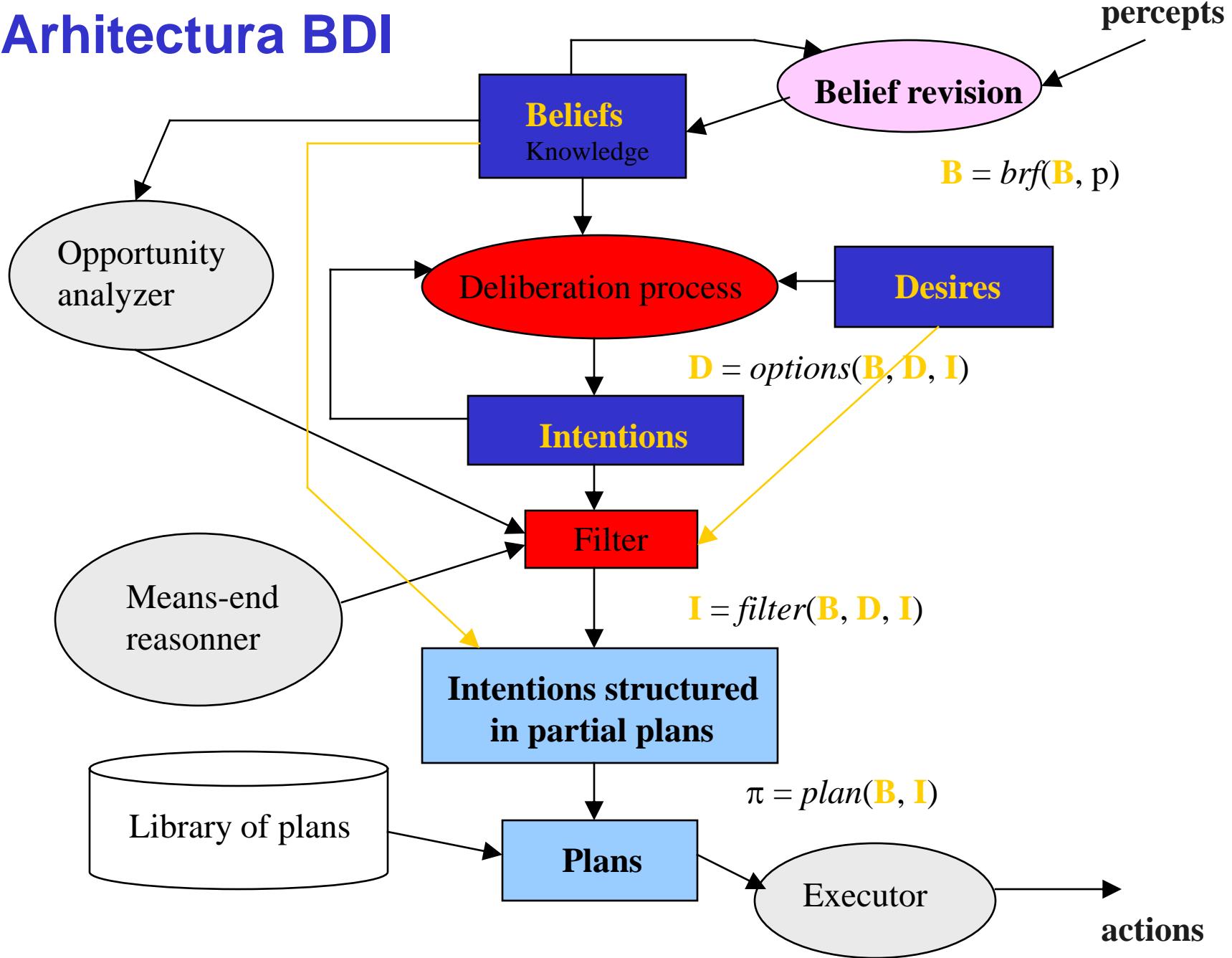
Masura a performantei

4.2 Arhitecturi de agenti cognitivi

Arhitecturi BDI

- Specificatii de nivel inalt
- **Beliefs (convingeri)** = informatii pe care agentul le are despre lume
- **Desires (dorinte)** = stari pe care agentul ar vrea sa le vada realizate
- **Intentions (intentii)** = dorinte (sau actiuni) pe care agentul s-a angajat sa le indeplineasca

Arhitectura BDI



Bucla de control a agentului

$$\mathbf{B} = \mathbf{B}_0 \quad \mathbf{I} = \mathbf{I}_0 \quad \mathbf{D} = \mathbf{D}_0$$

while true **do**

 get next percept p

$\mathbf{B} = \text{brf}(\mathbf{B}, p)$

$\mathbf{D} = \text{options}(\mathbf{B}, \mathbf{D}, \mathbf{I})$

$\mathbf{I} = \text{filter}(\mathbf{B}, \mathbf{D}, \mathbf{I})$

$\pi = \text{plan}(\mathbf{B}, \mathbf{I})$

 execute(π)

end while

Strategii de angajare (Commitment strategies)

- Optiune aleasa de agent ca intentie – **agentul s-a angajat pentru acea optiune**
- Persistenta intențiilor

Interbare: Cat timp se angajeaza un agent fata de o intenție?

- **Angajare oarba (Blind commitment)**
- **Angajare limitata (Single minded commitment)**
- **Angajare deschisa (Open minded commitment)**

B = B_0
I = I_0 **D** = D_0
while true **do**

Bucla de control BDI angajare oarba

get next percept p

B = **brf(B,p)**

D = **options(B, D, I)**

I = **filter(B, D, I)**

π = **plan(B, I)**

while not (empty(π) or succeeded (**I**, **B**)) **do**

α = head(π)

execute(α)

π = tail(π)

get next percept p

B = **brf(B,p)**

if not sound(π , **I, **B**) then**

π = **plan(B, I)**  *Reactivity, replan*

end while

end while

B = B_0
I = I_0 **D** = D_0
while true **do**

 get next percept p
 B = **brf(B,p)**
 D = **options(B, D, I)**
 I = **filter(B, D, I)**
 π = **plan(B, I)**

while not (empty(π) or succeeded (**I**, **B**) or impossible(**I**, **B**)) **do**
 α = head(π)
 execute(α)
 π = tail(π)
 get next percept p
 B = **brf(B,p)**
 if not sound(π , **I**, **B**) **then**
 π = **plan(B, I)** ← *Reactivity, replan*

end while
end while

Bucla de control BDI

angajare limitata

Dropping intentions that are impossible or have succeeded



B = B_0
I = I_0 **D** = D_0
while true **do**

Bucla de control BDI angajare deschisa

```
    get next percept p
    B = brf(B,p)
    D = options(B, D, I)
    I = filter(B, D, I)
     $\pi$  = plan(B, I)
    while not (empty( $\pi$ ) or succeeded (I, B) or impossible(I, B)) do
         $\alpha$  = head( $\pi$ )
        execute( $\alpha$ )
         $\pi$  = tail( $\pi$ )
        get next percept p
        B = brf(B,p)
        if reconsider(I, B) then
            |   D = options(B, D, I)
            |   I = filter(B, D, I)
             $\pi$  = plan(B, I)           ← Replan
    end while
end while
```

- Nu exista o unica arhitectura BDI
- PRS - Procedural Reasoning System (Georgeff)
- dMARS
- UMPRS si JAM – C++
- JACK – Java
- JADE - Java
- JADEX – XML si Java,
- JASON – Java

4.3 Arhitecturi de agenti reactivi

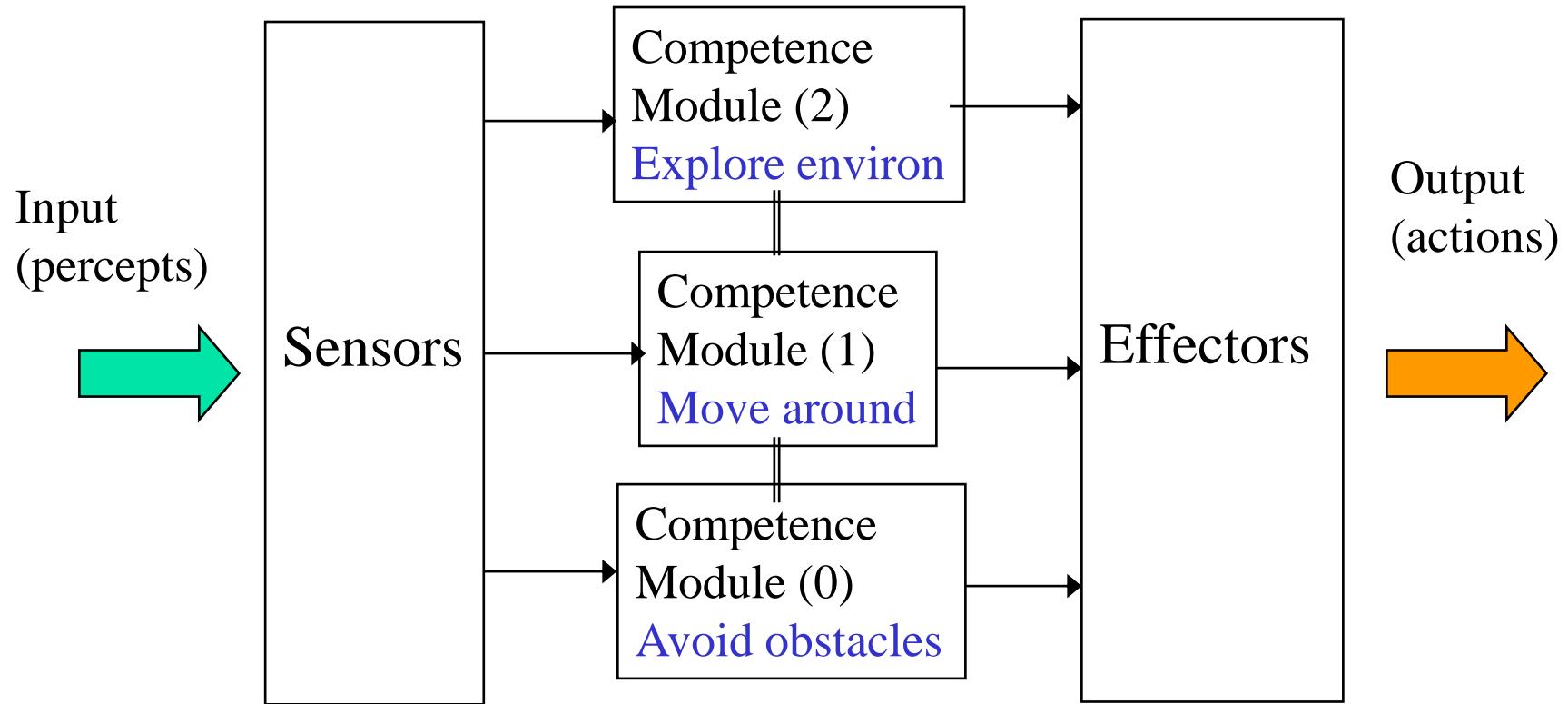
Arhitectura de subsumare - Brooks, 1986

- (1) Luarea deciziilor = { **Task Accomplishing Behaviours** }
 - Fiecare comportare (behaviour) = o functie ce realizeaza o actiune
 - Implementare: *situation → action*
- (2) Mai multe comportari pot fi activate in paralel

Arhitectura de subsumare

- Un TAB este reprezentat de un **modul de competenta** (c.m.)
- Fiecare c.m. executa un task simplu
- c.m. opereaza in paralel
- Nivele inferioare au prioritate fata de cele superioare
- c.m. la nivel inferior monitorizeaza si influenteaza intrarile si iesirile c.m. la nivel superior

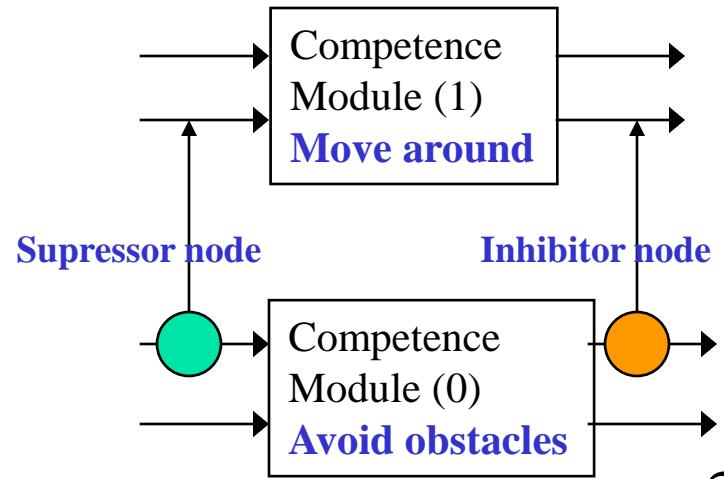
→ *subsumtion architecture*



$M_1 = \text{move around while avoiding obstacles} \supset M_0$

$M_2 = \text{explores the environment looking for distant objects of interests while moving around} \supset M_1$

- Incoroprarea functionalitatii unui c.m. subordonat de catre un c.m. superior se face prin noduri **supresoare** (modifica semnalul de intrare) si noduri **inhibitoare** (inhiba iesirea)



Comportare

(c, a) – conditie-actiune; descrie comportarea

$\mathbf{R} = \{ (c, a) \mid c \subseteq P, a \in A \}$ - multimea reguli de comportare

$\angle \subseteq R \times R$ – relatie binara totala de inhibare

function action(p: P)

var fired: P(R), selected: A

begin

fired = $\{(c, a) \mid (c, a) \in R \text{ and } p \in c\}$

for each (c, a) **in** fired **do**

if $\neg \exists (c', a') \in \text{fired}$ such that $(c', a') \angle (c, a)$ **then return** a

return null

end

Ne aflam pe o planeta necunoscuta care contine aur. Mostre de teren trebuie aduse la nava. Nu se stie daca sunt aur sau nu. Exista mai multi agenti autonomi care nu pot comunica intre ei. Nava transmite semnale radio: gradient al campului

Comportare

- (1) **Daca** detectez obstacol **atunci** schimb directia
- (2) **Daca** am mostre **si** sunt la baza **atunci** depune mostre
- (3) **Daca** am mostre **si** nu sunt la baza **atunci** urmez campul de gradient
- (4) **Daca** gasesc mostre **atunci** le iau
- (5) **Daca** adevarat **atunci** ma misc in mediu
 - (1) \angle (2) \angle (3) \angle (4) \angle (5)

Agentii pot comunica indirect:

- Depun si culeg boabe radiocative
- Pot seziza aceste boabe radioactive

- (1) **Daca** detectez obstacol **atunci** schimb directia
 - (2) **Daca** am mostre **si** sunt la baza **atunci** depune mostre
 - (3) **Daca** am mostre **si** nu sunt la baza **atunci** **depun** boaba **radioactiva** **si** urmez campul de gradient
 - (4) **Daca** gasesc mostre **atunci** le iau
 - (5) **Daca** gasesc boabe radioactive **atunci** iau una **si** urmez campul de gradient
 - (6) **Daca** adevarat **atunci** ma misc in mediul
-
- (1) < (2) < (3) < (4) < (5) < (6)

5. Negociere

5.1 Despre negociere

Agenti motivati colectiv = cooperare

Agenti motivati individual = competitie

Negociere = interactiune -> contract

- **Negocierea include:**

- un limbaj de comunicare
- un protocol de negociere
- un proces de decizie: concesii, criterii de acceptare/refuzare
- Single party **or** multi-party negotiation: one to many or many to many
(eBay <http://www.ebay.com>)

- **Tehnici de negociere**

- **Negociere bazata pe teoria jocurilor**
- **Negociere euristică**
- **Negociere bazata pe argumentare**

5.2 Negociere bazata pe teoria jocurilor

- Criterii de evaluare protocol negociere
- Agentii se comporta rational
- **Comportare rationala** = un agent prefera o utilitate / plata (*utility / payoff*) mai mare fata de una mai mica
- Functa de utilitate
 - $u_i: \Omega \rightarrow \mathbf{R}$
 - $\Omega = \{s_1, s_2, \dots\}$
 - $u_i(s) \geq u_i(s')$ ($s \geq s'$) – ordonarea preferintelor asupra rezultatelor

- Doi agenti au 2 actiuni posibile: **D** si **C** ($A_c = \{C, D\}$)
- Mediul se comporta astfel:

$$t: A_c \times A_c \rightarrow \Omega$$

$$t(D,D)=s_1 \quad t(D,C)=s_2 \quad t(C,D)=s_3 \quad t(C,C)=s_4$$

sau

$$t(D,D)=s_1 \quad t(D,C)=s_1 \quad t(C,D)=s_1 \quad t(C,C)=s_1$$

$$u_1(s_1)=4, \quad u_1(s_2)=4, \quad u_1(s_3)=1, \quad u_1(s_4)=1$$

$$u_2(s_1)=4, \quad u_2(s_2)=1, \quad u_2(s_3)=4, \quad u_2(s_4)=1$$

$$u_1(D,D)=4, \quad u_1(D,C)=4, \quad u_1(C,D)=1, \quad u_1(C,C)=1$$

$$u_2(D,D)=4, \quad u_2(D,C)=1, \quad u_2(C,D)=4, \quad u_2(C,C)=1$$

Agent1 $D,D \geq D,C \geq C,D \geq C,C$

- Agentii pot utiliza functii de utilitate pe care doresc sa le maximizeze
- Agentii trebuie sa aleaga acele actiuni care maximizeaza utilitatea (MEU)
- Daca sunt 2 sau mai multi agenti care fac actiuni care se conditioneaza reciproc – **teoria jocurilor**

Jocuri in forma normala

Forma normala a jocurilor

Un joc in forma normala cu **n** persoane este un tuplu (N, A, U) unde:

- N este o multime finita de **n** jucatori (indexati cu i)
- $A = A_1 \times \dots \times A_n$ unde A_i este multimea de actiuni a jucatorului i .
- Fiecare vector $a=(a_1, .., a_n) \in A$ se numeste **profil de actiune**
- $U=(u_1,..,u_n)$ unde $u_i:A \rightarrow R$ este functia de utilitate (payoff) a jucatorului i .
- Un mod natural de reprezentare este printr-o **matrice de n dimensiuni**

Exemple de jocuri in forma normala

Prisoner's dilemma

| Player A / Player B | Defect | Cooperate |
|------------------------|--------|-----------|
| Defect | 2 , 2 | 5 , 0 |
| Cooperate | 0 , 5 | 3 , 3 |

- Cooperate = nu marturiseste
- Defect = marturiseste

Jocuri cu castig comun

Jocul coordonarii

| | | Column player | |
|------------|-------|---------------|-------|
| | | Left | Right |
| Row player | Left | 1, 1 | 0, 0 |
| | Right | 0, 0 | 1, 1 |

- Un joc cu castig comun **common payoff game** este un joc in care pentru toate actiunile $\mathbf{a} \in A_1 \times \dots \times A_n$ si pentru orice pereche de agenti i, j exista egalitate $u_i(\mathbf{a}) = u_j(\mathbf{a})$
- Agentii nu au interese conflictuale, scopul lor este sa se coordoneze in actiuni pentru beneficiul maxim comun

Jocuri cu suma nula

Matching pennies

| | Heads | Tails |
|-------|-------|-------|
| Heads | 1, -1 | -1, 1 |
| Tails | -1, 1 | 1, -1 |

- Jocuri constante sau cu suma nula
- Un joc este un joc cu suma constanta (**constant sum game**) daca exista o constanta c atfel incat pentru fiecare strategie $a \in A_1 \times A_2$ exista relatia $u_1(a) + u_2(a) = c$
- Daca $c=0$ atunci **joc cu suma nula**
- Reprezinta competitie pura

Battle of sexes

| | Football | Movie |
|----------|----------|-------|
| Football | 2, 1 | 0, 0 |
| Movie | 0, 0 | 1, 2 |

- Jocuri care au atat elemente de cooperare cat si de competitie

Criterii in negociere

- **Comportare ratională** = utilitate (payoff) mai mare preferata fata de una mai mica
- **Maximizarea platii**: plata individuala, plata de grup, sau bunastare sociala
- **Bunastare sociala**
 - Suma utilitatii agentilor pentru o anumita situatie/solutie
 - Masoara binele general
 - Problema: cum compar utilitatile

Eficienta Pareto

■ Eficienta Pareto

- O solutie \mathbf{x} , i.e., un **vector de plata** $p(\mathbf{x}_1, \dots, \mathbf{x}_n)$, este **eficient Pareto**, i.e., Pareto optimal, daca nu exista alta solutie \mathbf{x}' a.i. cel putin un agent are o utilitate mai mare in \mathbf{x}' decat in \mathbf{x} si nici un agent nu are o utilitate mai mica in \mathbf{x}' decat in \mathbf{x} .
- Masoara bunastarea globala dar nu necesita compararea utilitatilor
- Bunastarea sociala \subset eficienta Pareto

■ Rationalitate individuala (IR)

- **IR a participarii unui agent** = Plata agentului in urma participarii la negociere nu este mai mica decat plata lui daca nu ar participa la negociere
- **O negociere este IR** daca este IR pentru toti agentii

Strategie dominanta

■ Stabilitate

- un protocol este **stabil** daca o data ce agentii au ajuns la o solutie ei nu deviaza de la aceasta
- **Strategie dominanta** = agentul are o utilitate mai mare folosind aceasta strategie indiferent de ce strategii folosesc ceilati agenti

$s_{ij} = t(\text{Act}_i, \text{Act}_j)$ rezultatul (starea) actiunilor Act_i a agentului i si Act_j a agentului j.

- O strategie $S_1 = \{s_{11}, s_{12}, \dots, s_{1n}\}$ **domina** o alta strategie $S_2 = \{s_{21}, s_{22}, \dots, s_{2n}\}$ daca orice rezultat $s_{ij} \in S_1$ este preferat (este mai bun) oricarui rezultat $s'_{ij} \in S_2$.

Strategie dominanta

- Daca exista o strategie dominanta agentul o urmeaza deoarece este garantat sa obtina cea mai mare utilitate

$$u_1(D,D)=4, \quad u_1(D,C)=4, \quad u_1(C,D)=1, \quad u_1(C,C)=1$$

$$u_2(D,D)=4, \quad u_2(D,C)=1, \quad u_2(C,D)=4, \quad u_2(C,C)=1$$

Agent1 $D, D \geq D, C \geq C, D \geq C, C$

- In acest caz agentul 1 are ca strategie dominanta D
- Agentul 2 nu are o strategie dominanta

Echilibru Nash

Echilibru Nash

- Doua strategii, S_1 a agentului A si S_2 a agentului B sunt in **echilibru Nash** :
 - daca agentul A urmeaza S_1 atunci agentul B nu poate obtine un castig mai mare decat acela obtinut daca urmeaza S_2 si
 - daca agent B urmeaza S_2 atunci agentul A nu poate obtine un castig mai mare decat acela obtinut daca urmeaza S_1 .
- Multime de strategii $\{S_1, \dots, S_k\}$ folosite de agentii A_1, \dots, A_k sunt in **echilibru Nash** daca, pentru orice agent A_i , strategia S_i este cea mai buna strategie a lui A_i daca ceilalți agenti folosesc $\{S_1, S_2, \dots, S_{i-1}, S_{i+1}, \dots, S_k\}$.

Probleme:

- nici un echilibru Nash
- multiple echilibre Nash

Prisoner's dilemma

| Player A / Player B | <i>Defect</i> | <i>Cooperate</i> |
|------------------------|---------------|------------------|
| <i>Defect</i> | 2 , 2 | 5 , 0 |
| <i>Cooperate</i> | 0 , 5 | 3 , 3 |

Bunastare sociala ?
Solutii Pareto ?
Echiliru Nash ?

Jocuri cu castig comun

Jocul coordonarii

| | | Column player | |
|------------|-------|---------------|-------|
| | | Left | Right |
| Row player | Left | 1, 1 | 0, 0 |
| | Right | 0, 0 | 1, 1 |

Jocuri care au atat elemente de cooperare cat si de competitie

Battle of sexes

| | Football | Movie |
|----------|----------|-------|
| Football | 2, 1 | 0, 0 |
| Movie | 0, 0 | 1, 2 |

Bunastare sociala ?

Solutii Pareto ?

Echiliru Nash ?

Game of Chicken

| | | Column player | |
|------------|------|---------------|------|
| | | Stay | Stir |
| Row player | Stay | 0, 0 | 3, 1 |
| | Stir | 1, 3 | 2, 2 |

Bunastare sociala ?

Solutii Pareto ?

Echiliru Nash ?

Jocuri cu suma nula

Matching pennies

| | Heads | Tails |
|-------|-------|-------|
| Heads | 1, -1 | -1, 1 |
| Tails | -1, 1 | 1, -1 |

Bunastare sociala ?

Solutii Pareto ?

Echiliru Nash ?

Jocuri cu suma nula

Matching pennies

| | Heads | Tails |
|-------|-------|-------|
| Heads | 1, -1 | -1, 1 |
| Tails | -1, 1 | 1, -1 |

- Daca P1 joaca Heads atunci P2 va juca Tails
- Dar acest lucru face ca P1 sa vrea sa joace Tails deasemenea
- Ceea ce face ca P2 sa vrea sa joace Heads
- etc. etc. etc.

Strategii in jocuri repetate

Turneul Axelrod

Strategii

- **ALL-D** – D tot timpul
- **RANDOM** – C sau D cu probabilitate egala
- **TIT-FOR-TAT**
 - C in primul tur
 - In turul $t > 1$ ce a ales oponentul in $t-1$
- **TESTER**
 - D in primul tur
 - Daca oponentul a ales D atunci TIT-FOR-TAT
 - Altfel joaca 2 tururi C si 1 tur D
- **JOSS**
 - TIT-FOR-TAT – dar cu 10% D

Strategii pure si mixte

- **Strategie pură:** selecteaza o singura actiune de executat
- **Strategie mixta:** selecteaza aleator dintre actiunile posibile cf unei distributii de probabilitate

Fie (N, A, U) un joc in forma normala
cu **profilul de strategie (actiune)** –

vectorul $s = (a_1, \dots, a_n) \in A$

Strategii mixte

(N,A,U) un joc in forma normala

- $A = A_1 \times \dots \times A_n - A_i$ setul de actiuni pentru un jucator i .

Fie (N,A,U) un joc in forma normala si, pentru orice multime X, fie $\mathbf{P}(X)$ multimea tuturor distributiilor de probabilitate peste X.

Multimea (profilul) de strategii mixte pentru jucatorul i este

$$S_i = \mathbf{P}(A_i)$$

Multimea profilelor de strategie este produsul cartezian a multimilor individuale de strategii mixte $S_1 \times \dots \times S_n$.

$s_i(a_i)$ indica probabilitatea ca o actiune a_i sa fie jucata cf strategiei mixte s_i .

Strategii mixte

Suportul unei strategii mixte s_i este

$$\{a_i \mid s_i(a_i) > 0\}$$

O strategie pură s_i este o strategie cu suport 1, i.e.

$$|\{a_i \mid s_i(a_i) > 0\}| = 1$$

O strategie pură joacă o singura actiune cu probabilitatea 1

Utilitatea asteptată pentru o strategie mixta

Fie (N, A, U) un joc în forma normală, utilitatea asteptată u_i a jucătorului i pentru profilul de strategie mixt $s = (s_1, \dots, s_n)$ se definește ca

$$u_i(s) = \sum_{a \in A} u_i(a) \prod_{j=1}^n s_j(a_j)$$

Echilibrul Nash pt strategii mixte

$$s_{-i} = (s_1, \dots, s_{i-1}, s_{i+1}, \dots, s_n)$$

Cel mai bun raspuns al jucatorului I la profilul de strategie s_{-i} este strategia (mixta) $s_i^* \in S_i$ pentru care

$$u_i(s_i^*, s_{-i}) \geq u_i(s_i, s_{-i})$$

pentru toate strategiile $s_i \in S_i$

Un profil de strategie $s = (s_1, \dots, s_n)$ este in **echilibru Nash** daca pentru toti agentii i , s_i este cel mai bun raspuns la s_{-i}

Profilul Nash este o strategie stabila: nici un agent nu si-ar schimba strategia chiar daca ar sti ce strategii joaca ceilalți

Matching Pennies

| | Heads | Tails |
|-------|-------|-------|
| Heads | 1, -1 | -1, 1 |
| Tails | -1, 1 | 1, -1 |

- Daca fiecare, privat, arunca cu banul atunci
 - Perechea de strategii $(1/2, 1/2)$ este in echilibru
- Avem deci un echilibru Nash cu strategie mixta

Matching Pennies

$$u_i(s) = \sum_{a \in A} u_i(a) \prod_{j=1}^n s_j(a_j)$$

| | Heads | Tails |
|-------|-------|-------|
| Heads | 1, -1 | -1, 1 |
| Tails | -1, 1 | 1, -1 |

$$U_{P1}(H)=U_{P1}(T)$$

$$1p + (-1)(1-p) = (-1)p + 1(1-p) \rightarrow \\ p=1/2$$

$$U_{P2}(H)=U_{P2}(T)$$

$$(-1)q + 1(1-q) = 1q + (-1)(1-q) \rightarrow \\ q=1/2$$

Alt exemplu strategie mixta

| P1 / P2 | A (x) | B (y) | C (1-x-y) |
|-----------|--------|-------|-----------|
| A (p) | 1, 1 | 10, 0 | -10, 1 |
| B (q) | 0, 10 | 1, 1 | 10, 1 |
| C (1-p-q) | 1, -10 | 1, 10 | 1, 1 |

- Jucatorul P1 joaca coloana A cu probabilitatea p , B cu probabilitatea q , si C cu probabilitatea $1-p-q$.
- P1 este indiferent intre A,B,C. Acelasi lucru pentru P2
- Este indiferent intre linia A si linia B daca x,y sunt astfel incat

$$1x + 10*y + (-10)*(1-x-y) = 0*x + 1*y + 10*(1-x-y).$$

- Este indiferent intre linia B si linia C daca ...

| P1 / P2 | A | B | C |
|---------|--------|-------|--------|
| A | 1, 1 | 10, 0 | -10, 1 |
| B | 0, 10 | 1, 1 | 10, 1 |
| C | 1, -10 | 1, 10 | 1, 1 |

- Construim profilul $(p,q; x,y)$ astfel incat celalalt jucator este indiferente fata de strategiile lui pure
- Astfel, indiferent de modul in care celalalt jucator joaca, utilitatea lui asteptata va fi cea din echilibru, adica $(p,q; x,y)$
- Nash a demonstrat ca orice joc in forma normala are un echilibru cu strategie mixta

6.3 Licitatii

- Protocole simple
- Centralizate
- Licitatii cu valoare privata
- Licitatii cu valoare comună
- Licitatii cu valoare corelata

Protocole de licitatii

- **English (first-price open cry) auction** – fiecare participant anunta deschis pretul pe care il liciteaza. Cel mai mare pret castiga
 - Strategie dominanta: putin mai mult decat ultimul pret, ma opresc cand ajung la valoarea privata – in licitatii cu valoare privata
 - In licitatii cu valoare corelata; creste constant pretul pana decizie stop
Winner's curse
- **First-price sealed-bid auction** – fiecare participant anunta pretul in plic inchis. Castiga cel cu pret maxim
 - Nu exista strategie dominanta; ofera cel mult pana la valoare lui privata

- **Dutch (descending) auction** - the auctioneer continuously lowers the price until one of the bidders takes the item at the current price.
 - Echivalenta cu licitatia *first-price sealed-bid*
- **Vickrey (second-price sealed-bid) auction** – trimite oferta in plic inchis. castiga cel care a facut cea mai mare oferta dar plateste al doilea pret
 - Strategia dominanta: valoarea lui private, nu este stimulat sa minta

Probleme in licitatii

6.4. Alocarea taskurilor prin negociere

- Alocare prin retea de contracte
 - Contract Net
 - Iterated Contract Net

Se afla intre negociere teoretica si euristică

FIPA - ACL

Foundation for Intelligent Physical Agents

- Scop FIPA = specificatii care permit interoperabilitatea intre sisteme multi-agent
- Vede mesajele ca acte de comunicare (CA)
- Foloseste 20 CAs
- Are un limbaj cu o semnatica formala – SL (Semantic Language)

FIPA communicative acts

Informatives

- query_if, subscribe, **inform**, inform_if, confirm, disconfirm, not_understood

Task distribution

- **request**, request Whenever, cancel, agree, refuse, failure

Negotiation

- cfp, propose, accept_proposal, reject_proposal

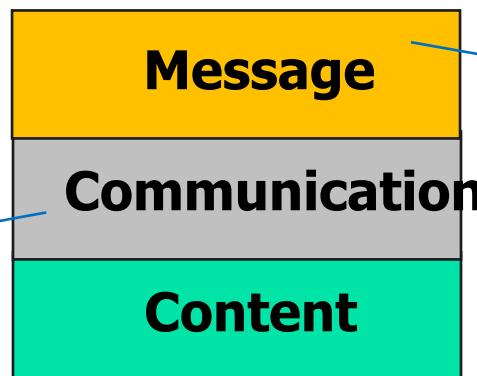
FIPA ACL

separa:

- Semantica comunicarii (independenta de domeniu) - **message**
- Semantica mesajului (dependentă de domeniu) - **content**

Un mesaj FIPA are 3 niveluri conceptuale

Descrie parametrii comunicarii de nivel jos:
- Identitate emitor si receptor
- Un id unic asociat comunicarii



Core al FIPA
- **Communication primitive (speech act)**
Optional
- **content language**
- **ontology**

FIPA-SL

```
(request  :sender (agent-identifier :name i)
         :receiver (set (agent-identifier :name j)
                      :content ((action (agent-identifier :name j)
                                       (deliver box7 (loc 10 15))))
                     :protocol fipa-request
                     :language fipa-sl
                     :reply-with order56 )

(agree   :sender (agent-identifier :name j)
         :receiver (set (agent-identifier :name i)
                      :content ((action (agent-identifier :name j)
                                       (deliver box7 (loc 10 15))) (priority order56 low))
                     :protocol fipa-request
                     :language fipa-sl
                     :in-reply-to order56 )
```

Exemplu

(cfp

Agentul *j* ii cere lui *i* / propuneri de vanzare
a 50 cutii de prune si conditii de pret

:sender (agent-identifier :name j)

:receiver (set (agent-identifier :name i))

:content

"((action (agent-identifier :name i)

(sell plum 50))

(any ?x (and (= (price plum) ?x) (< ?x 10))))"

:ontology fruit-market

:language fipa-sl)

Exemplu

(propose

Agentul *j* propune lui *i* sa-i vanda
50 cutii prune la pret de 5

:sender (agent-identifier :name j)

:receiver (set (agent-identifier :name i))

:content

"((action j (sell plum 50))

(= (any ?x (and (= (price plum) ?x) (< ?x 10))) 5)"

:ontology fruit-market

:in-reply-to proposal2

:language fipa-sl)

Exemplu

(accept-proposal Agentul *i* acceptă pe *j*)
:sender (agent-identifier :name i)
:receiver (set (agent-identifier :name j))
:in-reply-to bid089
:content
" ((action (agent-identifier :name j)
 (sell plum 50))
 (= (price plum) 5))) "
:language fipa-sl)

Exemplu

(reject-proposal

Agentul *i* refuza pe *k*

:sender (agent-identifier :name *i*)

:receiver (set (agent-identifier :name *k*))

:content

"((action (agent-identifier :name *k*)

 (sell plum 50))

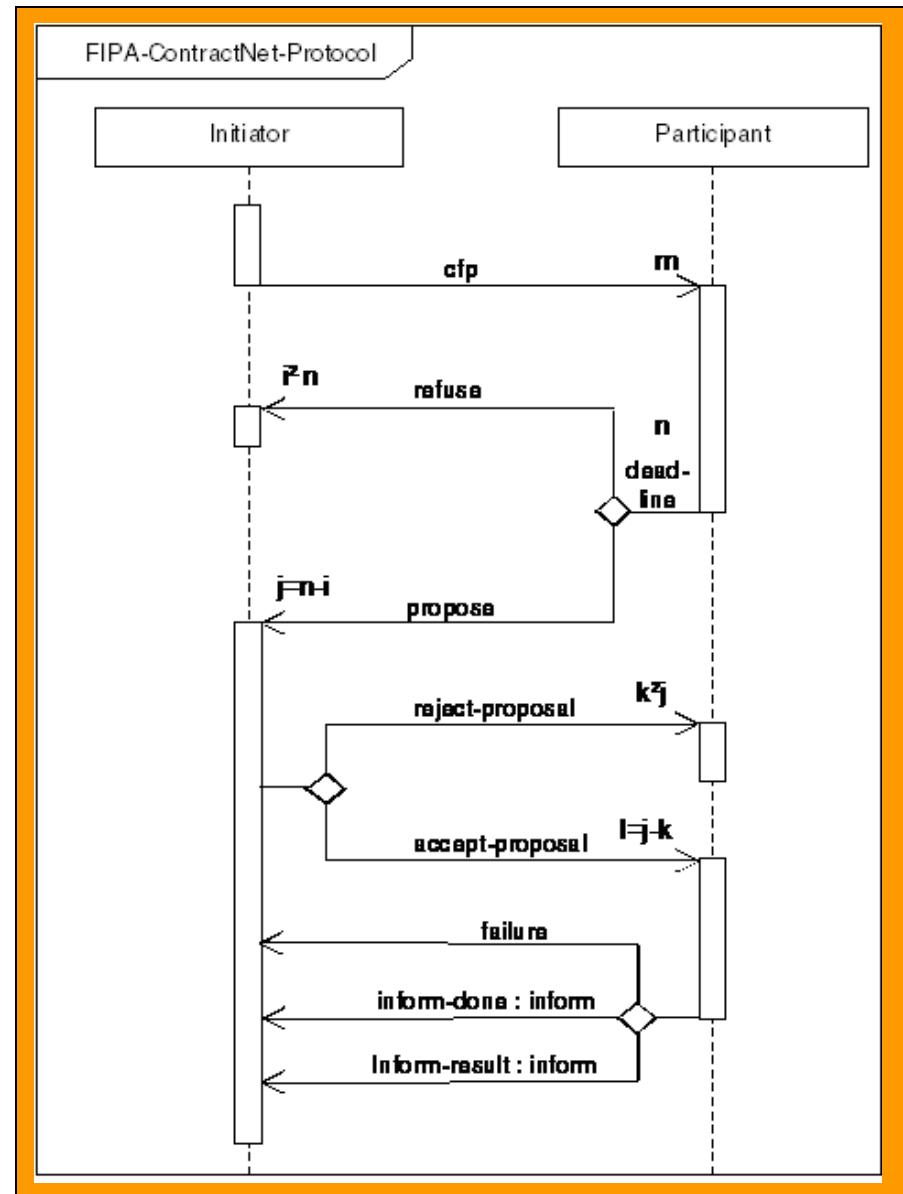
 (= (price plum) 20)

 (price-too-high 20))"

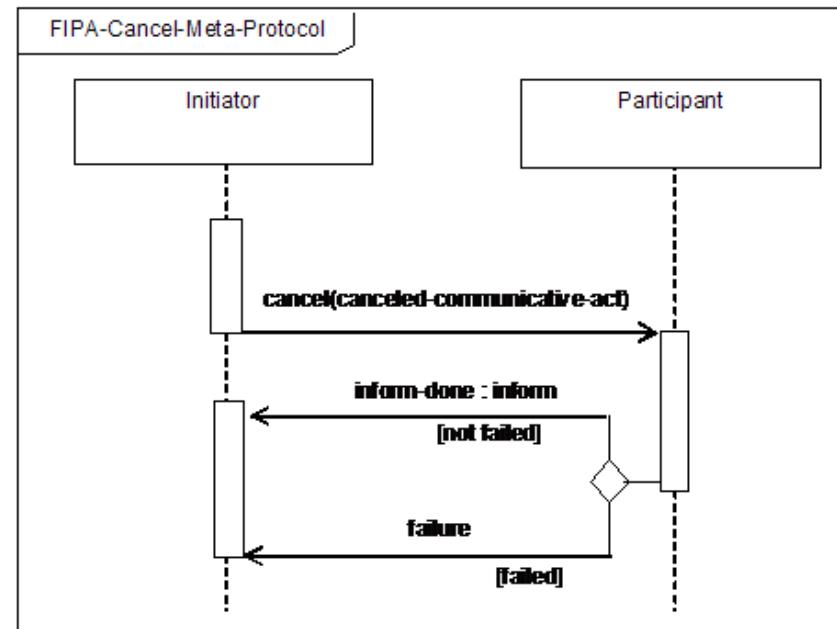
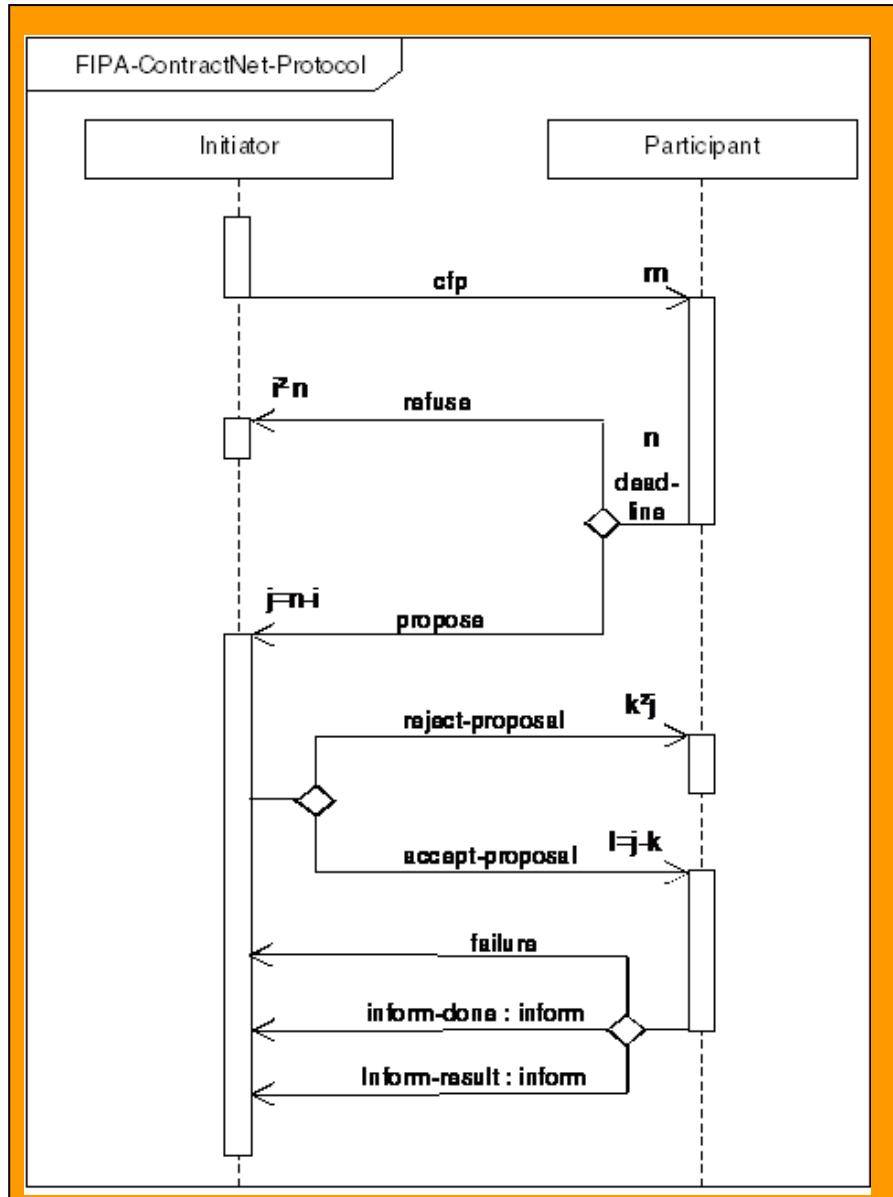
:in-reply-to bid080)

FIPA - Contract net

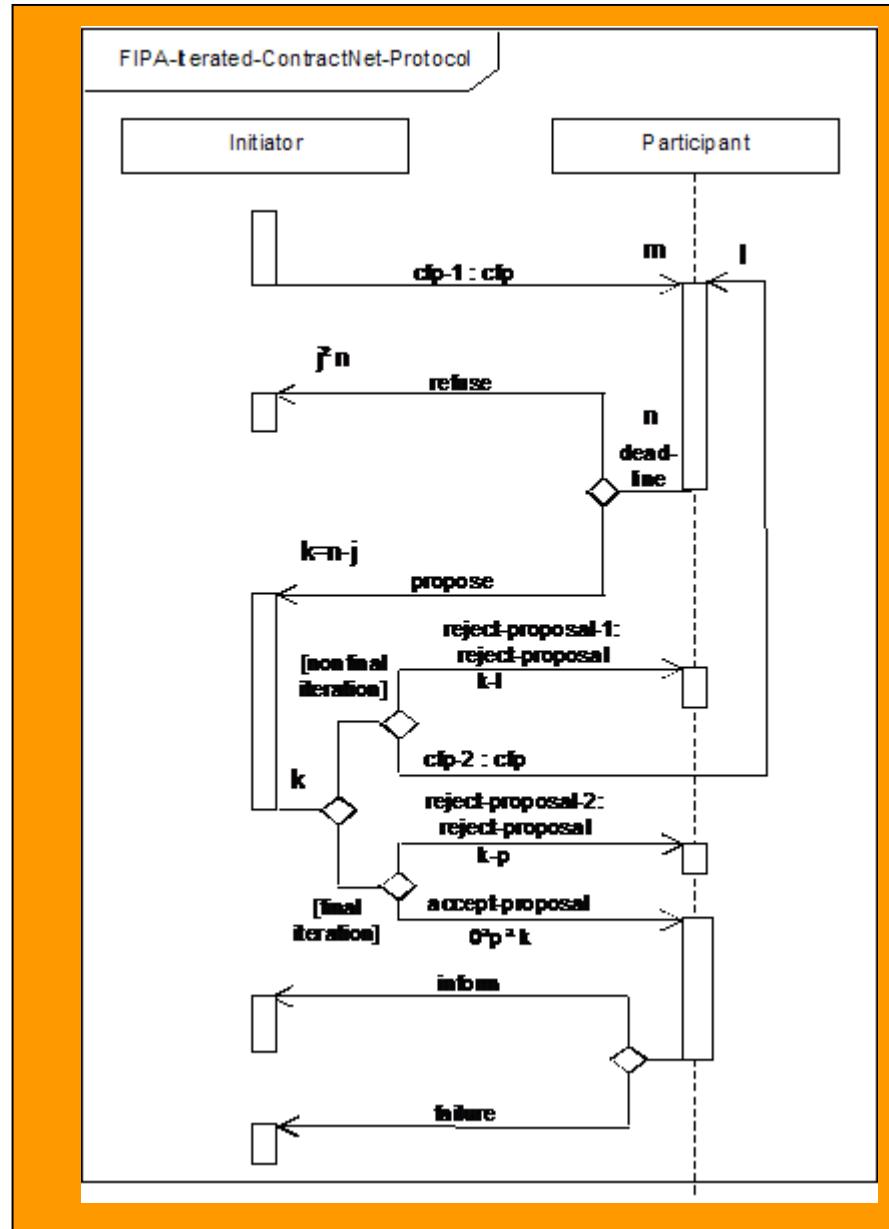
- Initiatorul solicita propuneri –
cfp: specifica taskul si conditii asupra lui, de la n participanti (contractori)
 - Cei n participanti genereaza raspunsuri:
 - i refuza (**refuse**)
 - j=n-i propun (**propose**), inclusiv conditii
 - cfp include un deadline pt raspunsuri (apoi sunt automat exclude)
 - Initiatorul evaluateaza propunerile si selecteaza l (intre 1 si j) agenti – li se trimit mesaj de **accept-proposal** si mesaj de **reject-proposal** la k=j-l agenti
 - Propunerile angajeaza participantii
 - Un participant trebuie sa raspunda cu:
 - **inform-done** sau
 - **inform-result** (daca a executat taskul) sau
 - **failure**.
- Interactiunea este identificata printr-un unic parametru *conversation-id*



FIPA - Contract net



FIPA – Iterated Contract net





Inteligentă Artificială

Universitatea Politehnica Bucuresti
Anul universitar 2020-2021

Adina Magda Florea



Curs nr. 11

Prelucrarea limbajului natural

- Prelucrare LN pentru comunicare
- Prelucrare LN pt achizitia cunostintelor

Prelucrare LN pentru comunicare

1 Comunicare

Acte de comunicare

J. Austin - *How to do things with words*, 1962, J. Searle - *Speech acts*, 1969

Un act de comunicare:

- **locutie** = fraza spusa de locutor
- **illocutie** = intelestul dorit spre a fi comunicat de locutor (performativa)
- **prelocutie** = actiunea care rezulta din locutie

Maria i-a spus lui Ion: "Te rog inchide usa"

locutie

illocutie continut

prelocutie: **usa inchisa**

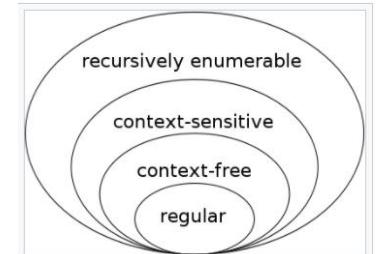
Categorii ilocationale

- **Asertive**
- **Directive**
- **Permissive**
- **Prohibitive**
- **Declarative**
- **Expressive**

2 Modele limbaj

- Modelele n-gram - bazate pe secvențe de caractere, cuvinte
- Modele de limbaj bazate pe structura gramaticală
- Categorii lexicale (parts of speech): substantiv, adjecțiv, etc
- Categorii sintactice: grup substantival, grup verbal, etc.
- Structuri de fraze

Modele limbaj



| Gramatica / Clasa limbaj | Automat | Reguli de productie |
|--------------------------|--------------------------------|--|
| Recursiv numarabile | Masina Turing | $\gamma \rightarrow \alpha$ |
| Dependente de context | Masina Turing linear marginita | $\alpha A \beta \rightarrow \alpha \gamma \beta$ |
| Independente de context | Automate Push down | $A \rightarrow \alpha$ |
| Regulate | Automate cu stari finite | $A \rightarrow a$ $A \rightarrow aB$ |

Fernando Pereira: "*The older I get, the further down the Chomsky hierarchy I go*"

Definire limbaj

- Lexicon, categorii deschise si inchise
- Analiza lexicala
- Analiza sintactica (pars oratoris)
- Gramatici
- Terminale, neterminale
- Reguli de rescriere/productii ($LHS \rightarrow RHS$)
- Analiza semantica + analiza pragmatica

3 Analiza lexicala

Gramatici regulate

$$G = (N, \Sigma, R, S)$$

N este multimea de neterminale

Σ este multimea de terminale

R este multimea de reguli de rescriere

$$R: \{ X \rightarrow a \text{ sau } X \rightarrow aY \mid X, Y \in N \text{ si } a \in \Sigma \})$$

S – simbolul de start

- Genereaza Lexiconul

4 Analiza sintactica

Gramatici independente de context

$$G = (N, \Sigma, R, S)$$

N este multimea de neterminale

Σ este multimea de terminale

R este multimea de reguli de rescriere

$$R: N \rightarrow (N \cup \Sigma)^*$$

S – simbolul de start

Gramatica pt sintaxa

$S \rightarrow NP VP$
 $S \rightarrow Aux NP VP$
 $S \rightarrow VP$
 $NP \rightarrow Pronoun$
 $NP \rightarrow Proper-Noun$
 $NP \rightarrow Det Nominal$
 $Nominal \rightarrow Noun$
 $Nominal \rightarrow Nominal Noun$
 $Nominal \rightarrow Nominal PP$
 $VP \rightarrow Verb$
 $VP \rightarrow Verb NP$
 $VP \rightarrow Verb NP PP$
 $VP \rightarrow Verb PP$
 $VP \rightarrow VP PP$
 $PP \rightarrow Preposition NP$

Lexicon

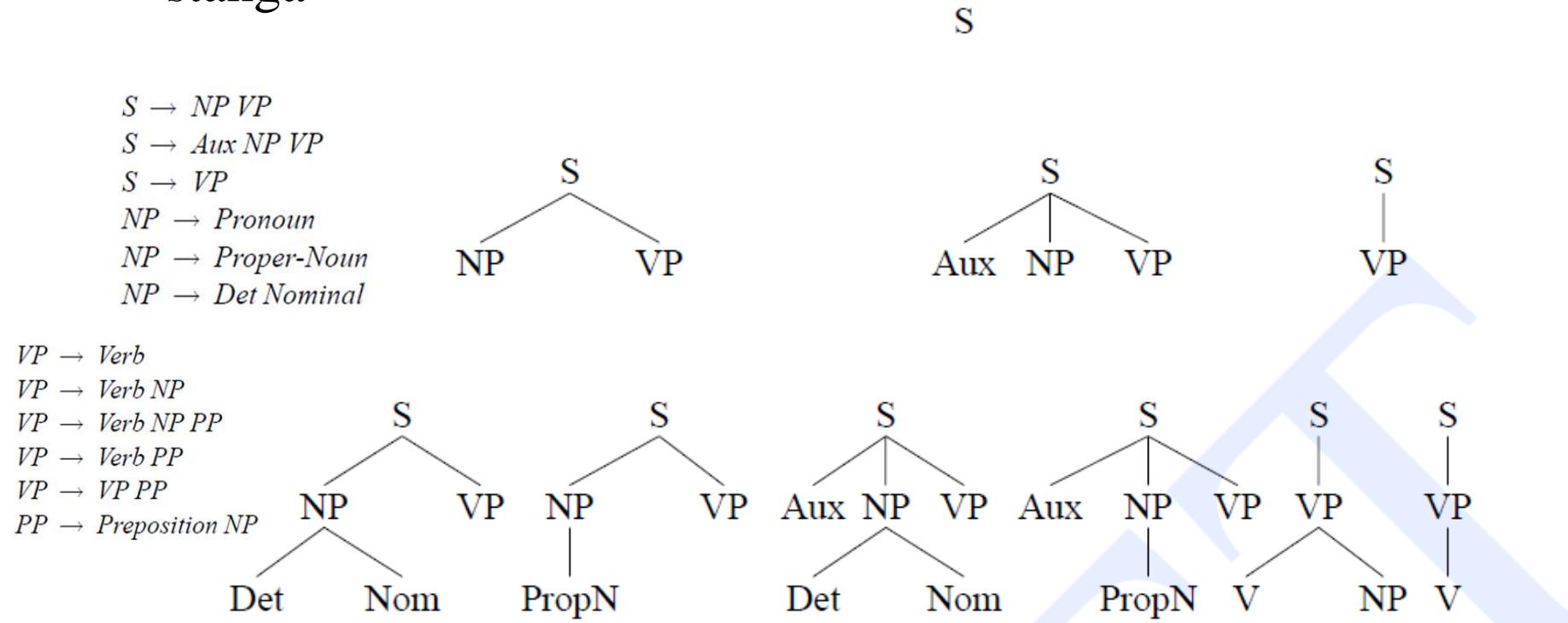
$Det \rightarrow that | this | a$
 $Noun \rightarrow book | flight | meal | money$
 $Verb \rightarrow book | include | prefer$
 $Pronoun \rightarrow I | she | me$
 $Proper-Noun \rightarrow Houston | TWA$
 $Aux \rightarrow does$
 $Preposition \rightarrow from | to | on | near | through$

Care este arborele de derivare/sintactic al frazei **Book that flight?**

Analiza Top-down

Book that flight

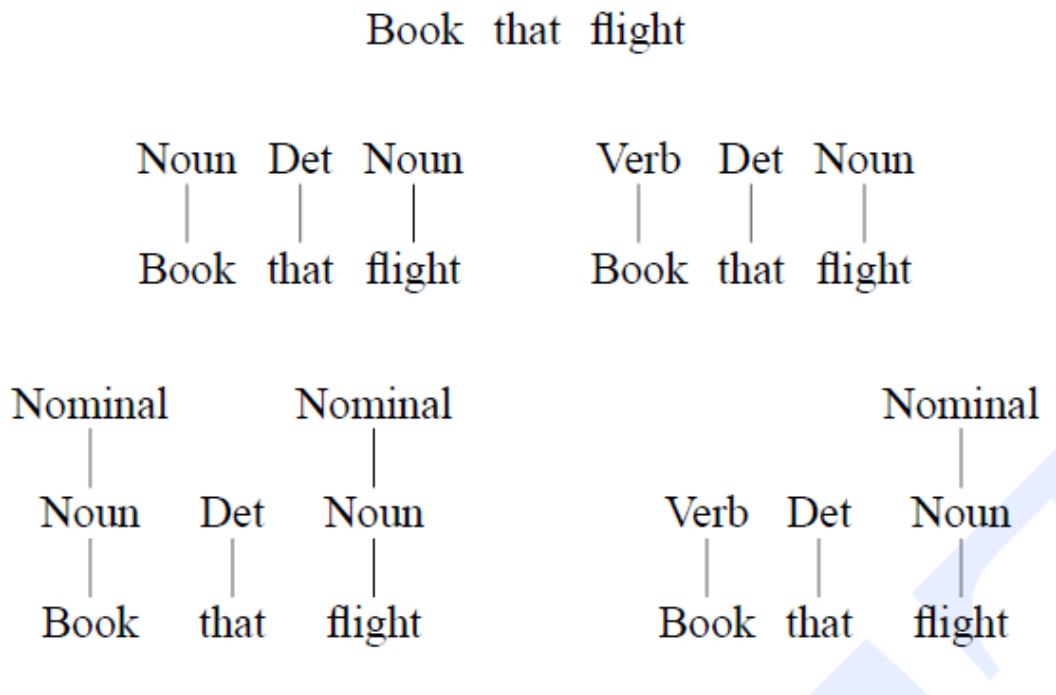
- Analiza top-down cauta arborele de derivare incepand de la S catre frunze
- Pe fiecare nivel se expandeaza neterminalul cel mai din stanga



Analiza Bottom-Up

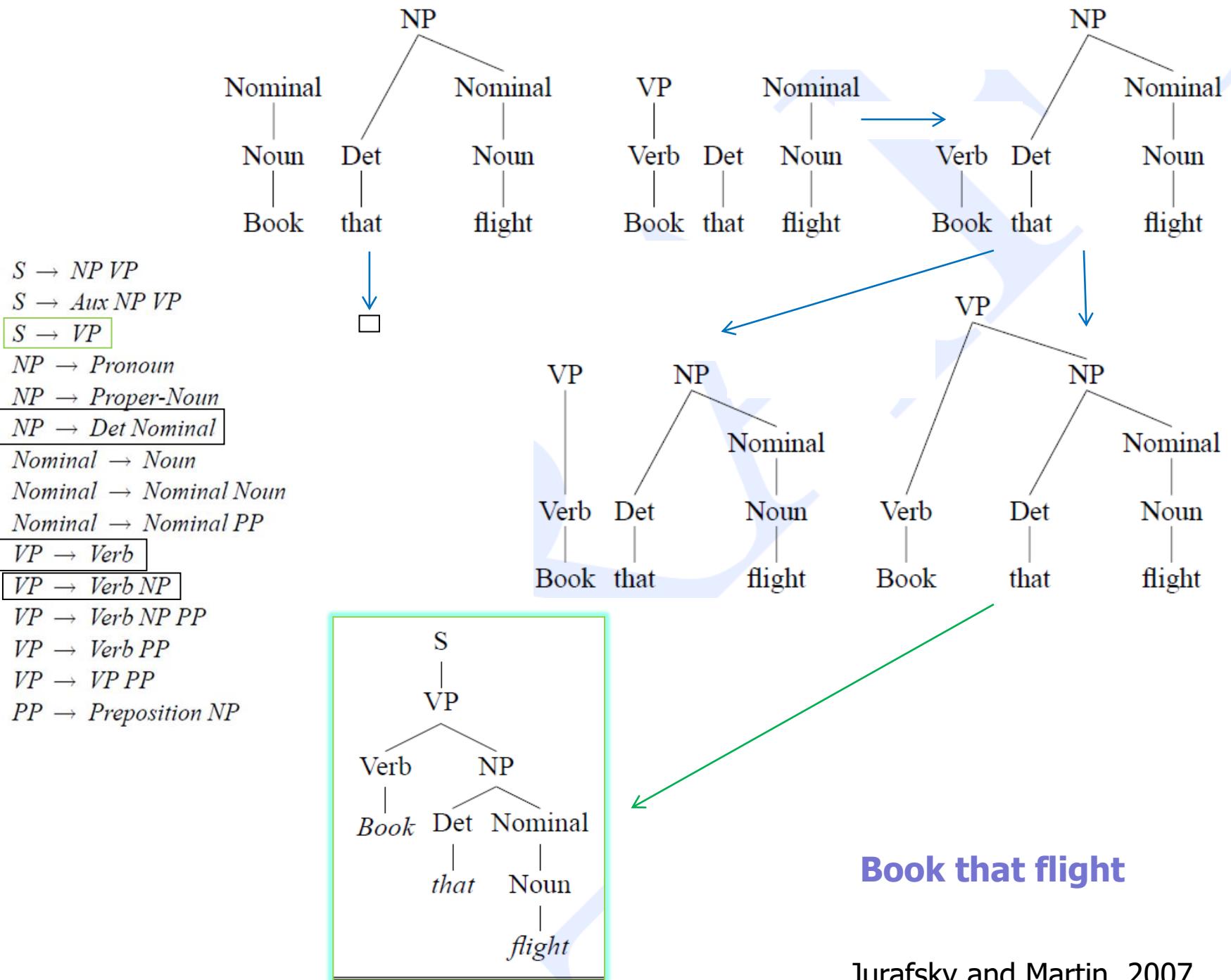
Book that flight

- Porneste de la cuvintele din fraza si cauta sa obtina S
- Folosita si la analizoarele “shift-reduce” pt limbaje de programare



$\boxed{\text{Det} \rightarrow \text{that} \mid \text{this} \mid \text{a}}$
 $\boxed{\text{Noun} \rightarrow \text{book} \mid \text{flight} \mid \text{meal} \mid \text{money}}$
 $\boxed{\text{Verb} \rightarrow \text{book} \mid \text{include} \mid \text{prefer}}$

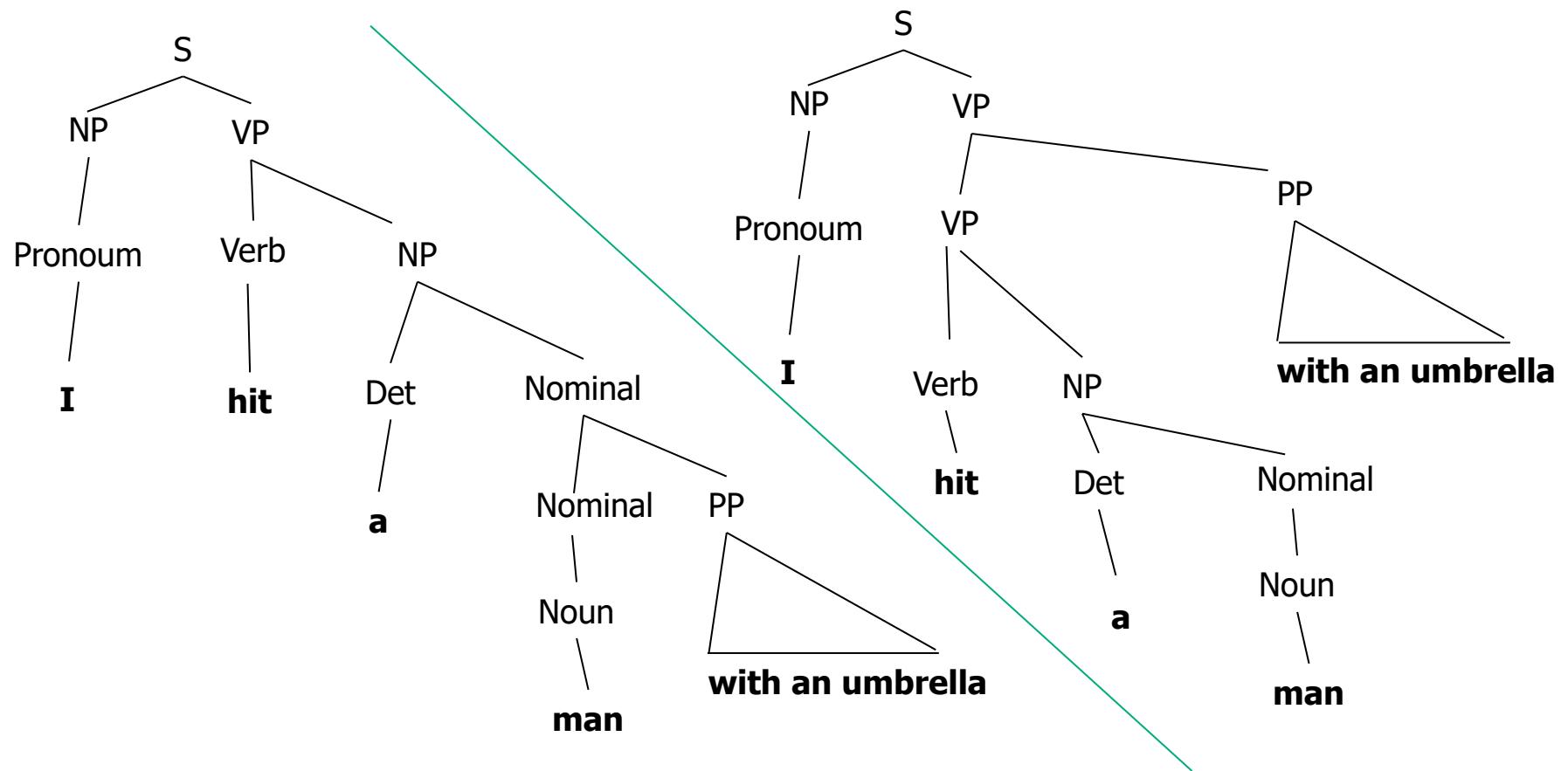
$S \rightarrow NP VP$
 $S \rightarrow Aux NP VP$
 $S \rightarrow VP$
 $NP \rightarrow \text{Pronoun}$
 $NP \rightarrow \text{Proper-Noun}$
 $NP \rightarrow \text{Det Nominal}$
 $\boxed{Nominal \rightarrow Noun}$
 $Nominal \rightarrow \text{Nominal Noun}$
 $Nominal \rightarrow \text{Nominal PP}$
 $VP \rightarrow \text{Verb}$
 $VP \rightarrow \text{Verb NP}$
 $VP \rightarrow \text{Verb NP PP}$
 $VP \rightarrow \text{Verb PP}$
 $VP \rightarrow \text{VP PP}$
 $PP \rightarrow \text{Preposition NP}$



Book that flight

Problema ambiguitatii sintactice

- Ambiguitate sintactica sau **ambiguitate structurala**
- Gramatica genereaza mai multi arbori de analiza



Problema ambiguitatii sintactice

Ambiguitate structurala

- **Ambiguitate de asociere** (attachment ambiguity) – un constituent poate fi atasat in arborele de derivare in mai mult de 1 loc
 - I hit a man with an umbrella
 - We saw the Eiffel Tower flying over Paris
- **Ambiguitate de coordonare** (coordination ambiguity) - parti de fraza care pot fi cuplate diferit de conjunctii cum ar fi “and”
 - Old men and women
 - He speaks good French and English

Analiza sintactica pt LN

- Ambiguitatea structurala poate genera multi arbori sintactici
(Old men and women speak good French and English)
- Pentru anumite gramatici, numarul posibil de arbori poate fi exponential in raport cu numarul de cuvinte din fraza
- O solutie – Gramaticile independente de context probabiliste

GICP (PCFG)

- Gramatici independente de context probabiliste (GICP)
 - VP → Verb [0.70]
 - | Verb NP [0.30]
- Determinare probabilitati
 - de proiectant
 - pe baza de treebanks (fraze deja anlizate corect), de ex. Penn Treebank (<https://en.wikipedia.org/wiki/Treebank>)

Gramatici regulate probabiliste

- Lexicon

Noun → **breeze** [0.10] | **wumpus** [0.15] | **ball** [0.15] ...

Verb → **is** [0.10] | **see** [0.10] | **smells** [0.10] | **hit** [0.10] ...

Adjective → **right** [0.10] | **left** [0.10] | **smelly** [0.15] ...

Adverb → **here** [0.05] | **there** [0.05] | **ahead** [0.02] ...

Pronoun → **me** [0.10] | **you** [0.03] | **I** [0.10] | **it** [0.10] ...

RelPronoun → **that** [0.40] | **who** [0.20] ...

Name → **John** [0.1] | **Mary** [0.01] ...

Article → **the** [0.40] | **a** [0.30] | **an** [0.10] ...

Preposition → **to** [0.20] | **in** [0.10] | **on** [0.05] ...

Conjunction → **and** [0.50] | **or** [0.10] | **but** [0.20] ...

GIC probabiliste - sintaxa

- Sintaxa

| | | |
|------------------------|--------|-------------------------------|
| $S \rightarrow NP\ VP$ | [0.90] | I feel a breeze |
| S Conjunction S | [0.10] | I feel a breeze and it stinks |

| | | | |
|------------------|---------|--------|---------------------------------|
| $NP \rightarrow$ | Pronoun | [0.30] | I |
| Name | | [0.10] | John |
| Noun | [0.10] | [0.10] | pit |
| Article Noun | | [0.25] | the wumpus |
| NP PP | | [0.10] | the wumpus in 1,3 |
| NP RelClause | | [0.05] | the wumpus that is smelly |

| | | | |
|------------------|----------------|--------|---------------|
| $VP \rightarrow$ | Verb | [0.40] | stinks |
| VP NP | | [0.35] | feel a breeze |
| VP Adjective | | [0.05] | smells dead |
| VP PP | | [0.10] | is in 1,3 |
| VP Adverb | | [0.10] | go ahead |
| $PP \rightarrow$ | Preposition NP | [1.00] | to the east |

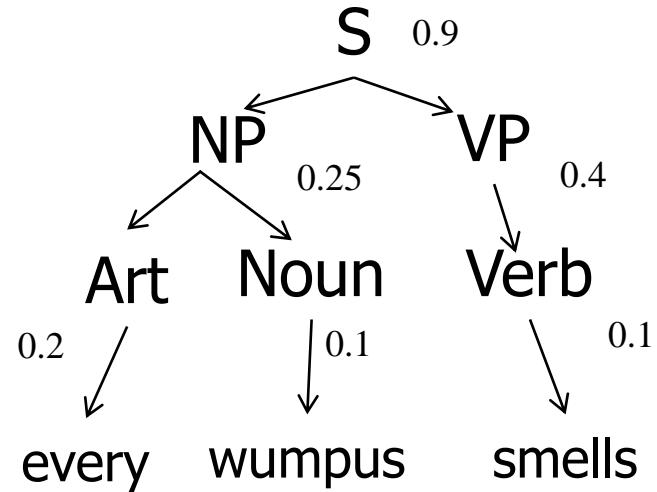
$RelClause \rightarrow RelPronoun\ VP$ [1.00] that is smelly

Analiza sintactica cu GICP

- Fiecare arbore de derivare intr-o GICP are asociata o probabilitate $P(t)$

Ex: $0.9 \times 0.25 \times 0.2 \times 0.1 \times 0.4 \times 0.1 = 0.00018$

| | |
|---|--------|
| $S \rightarrow NP \ VP$ | [0.90] |
| $NP \rightarrow Art \ Noun$ | [0.25] |
| $Art \rightarrow \text{every}$ | [0.2] |
| $Noun \rightarrow \text{wumpus}$ | [0.1] |
| $VP \rightarrow \text{Verb}$ | [0.4] |
| $\text{Verb} \rightarrow \text{smells}$ | [0.1] |



Analiza sintactica cu GICP

- Fiind data o GICP si o fraza s , fie $T(s)$ multimea de arbori de derivare asociati lui s
- Ne intereseaza sa gasim

$$\arg \max_{t \in T(s)} P(t)$$

Avem 2 posibilitati:

- 1) **Generez toti arborii posibili**; calculez probabilitatile asociate si aleg arborele cu cea mai mare probabilitate

Ineficienta datorita numarului mare de arbori posibili

- 2) **Bazata pe programare dinamica / chart parsers**

Analizez subsiruri si memorez analizele partiale

Analiza CKY (Cocke–Younger–Kasami)

- Analiza CKY este o analiza “bottom-up” în care după ce analizăm un subsir, se memorează rezultatul într-o tabelă (**chart**), astfel încât să il putem reutiliza - **chart parser**
- GIC – orice subsir/fraza analizată pe o ramură a arborelui sintactic poate fi utilizată pe alta ramură
- Chart parser-ul **CKY** (J. Cocke, D. Young, T. Kasami)
- Gramatica trebuie să fie în **Forma Normală Chomsky (CNF)**; orice regulă trebuie să aibă una din două forme:
 - $X \rightarrow a$ cu $a \in \Sigma$ (reguli lexicale)
 - $X \rightarrow Y Z$ cu $Y, Z \in N$ (reguli sintactice)

Transformarea in Chomsky Normal Form

Trebuie sa transformam urmatoare tipuri de reguli:

- Reguli care combina terminale cu neterminale

$X \rightarrow a Y [p]$ transform $X \rightarrow A Y [p]$ si $A \rightarrow a [1.0]$

- Reguli care au 1 singur neterminal in partea dreapta (unitare)

Se rescrie partea dreapta cu partea dreapta a tuturor regulilor neunitare

$X \rightarrow Y [p]$ si $Y \rightarrow a [p1]$ transform

$X \rightarrow a [p \times p1]$

Transformarea in CNF

- Reguli care au mai mult de 2 neterminale in dreapta

Introduc noi neterminale

$X \rightarrow Y Z T [p]$ transform in

$X_1 \rightarrow Y Z [p]$

$X \rightarrow X_1 T [1.0]$

- Orice GIC (nu neaparat GICP) poate fi astfel transformata in CNF

Analiza sintactica

- CKY pt GICP foloseste un spatiu de $O(N^2M)$, N – nr cuvinte (lungimea sirului de intrare), M – nr neterminale din gramatica, pentru a construi o **tabela de probabilitati P**
- Timp $O(N^3M)$
- Nu examineaza toti a.d. posibili ci calculeaza probabilitatea celui mai probabil a.d.
- Toti subarborii sunt implicit reprezentati in tabela P de unde se pot obtine daca dorim

Exemplul 1

The flight includes a meal

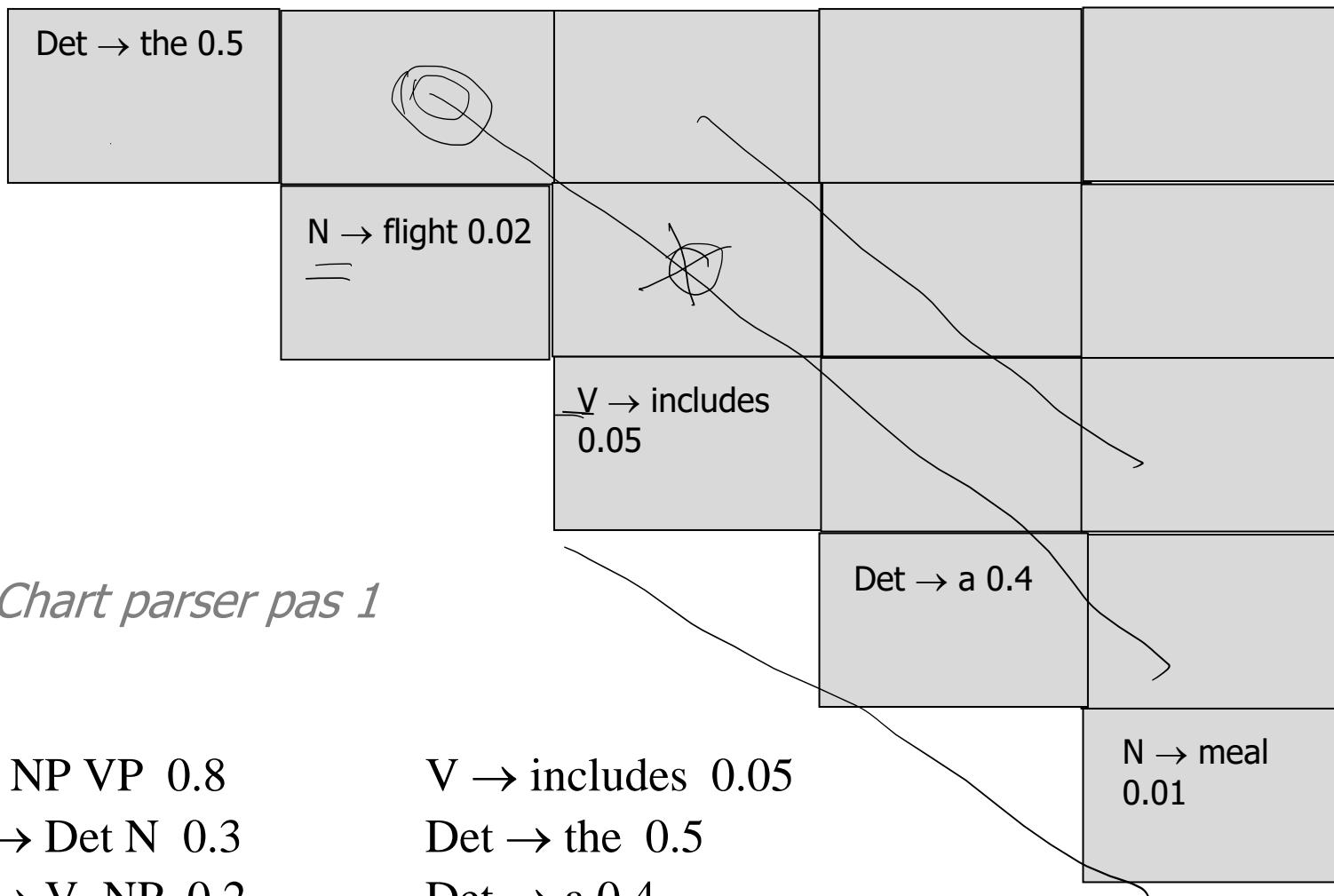


Chart parser pas 1

$S \rightarrow NP\ VP \ 0.8$
 $NP \rightarrow Det\ N \ 0.3$
 $VP \rightarrow V\ NP \ 0.2$

$V \rightarrow includes \ 0.05$
 $Det \rightarrow the \ 0.5$
 $Det \rightarrow a \ 0.4$
 $N \rightarrow meal \ 0.01$
 $N \rightarrow flight \ 0.02$

Exemplul 1

The flight includes a meal

| | | | | |
|---------------|--|-------------|---|--|
| Det → the 0.5 | NP → Det N $0.3 \times 0.5 \times 0.02 = 0.003$ | | | |
| | N → flight 0.02 | | | |
| | V → includes 0.05 | | | |
| | | Det → a 0.4 | NP → Det N $0.3 \times 0.4 \times 0.01 = 0.0012$ | |
| | | | N → meal 0.01 | |

Chart parser pas 2

$S \rightarrow NP\ VP\ 0.8$
 $NP \rightarrow Det\ N\ 0.3$
 $VP \rightarrow V\ NP\ 0.2$

$V \rightarrow includes\ 0.05$
 $Det \rightarrow the\ 0.5$
 $Det \rightarrow a\ 0.4$
 $N \rightarrow meal\ 0.01$
 $N \rightarrow flight\ 0.02$

Exemplul 1

The flight includes a meal

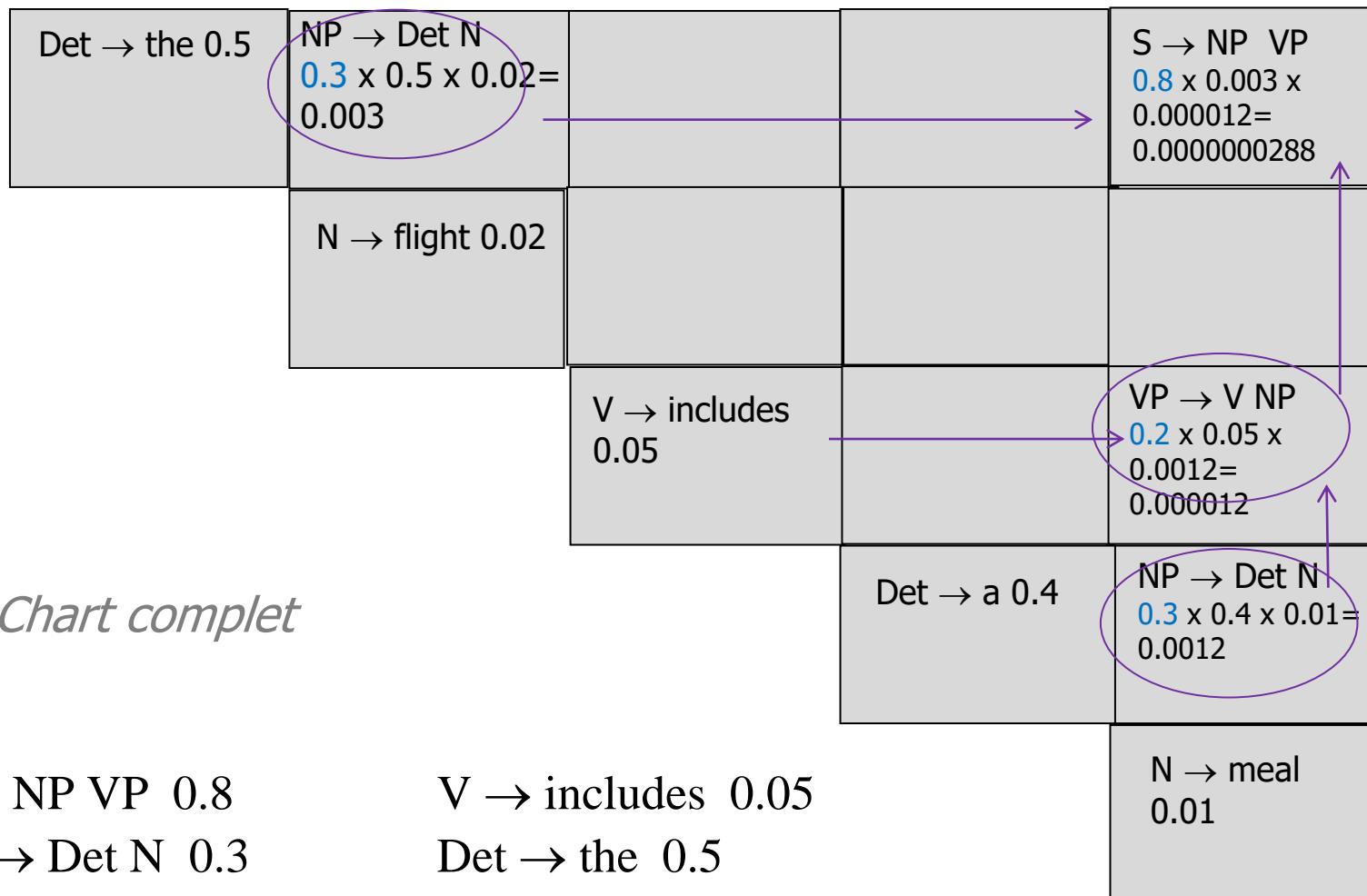


Chart complet

$$S \rightarrow NP VP \quad 0.8$$

$$NP \rightarrow Det N \quad 0.3$$

$$VP \rightarrow V NP \quad 0.2$$

$$V \rightarrow includes \quad 0.05$$

$$Det \rightarrow the \quad 0.5$$

$$Det \rightarrow a \quad 0.4$$

$$N \rightarrow meal \quad 0.01$$

$$N \rightarrow flight \quad 0.02$$

Exemplul 2

| G initială | | Fish people fish tanks | |
|------------------------|-----|-------------------------|-------|
| | | G transformata in CNF | |
| $S \rightarrow NP VP$ | 0.9 | $S \rightarrow NP VP$ | 0.9 |
| $S \rightarrow VP$ | 0.1 | $S \rightarrow V NP$ | 0.05 |
| | | $S \rightarrow people$ | 0.001 |
| | | $S \rightarrow fish$ | 0.006 |
| $VP \rightarrow V NP$ | 0.5 | $VP \rightarrow V NP$ | 0.5 |
| $VP \rightarrow V$ | 0.1 | $VP \rightarrow people$ | 0.01 |
| | | $VP \rightarrow fish$ | 0.06 |
| $NP \rightarrow NP NP$ | 0.2 | $NP \rightarrow NP NP$ | 0.2 |
| $NP \rightarrow N$ | 0.7 | $NP \rightarrow people$ | 0.35 |
| | | $NP \rightarrow fish$ | 0.14 |
| | | $NP \rightarrow tanks$ | 0.14 |
| $N \rightarrow people$ | 0.5 | $N \rightarrow people$ | 0.5 |
| $N \rightarrow fish$ | 0.2 | $N \rightarrow fish$ | 0.2 |
| $N \rightarrow tanks$ | 0.2 | $N \rightarrow tanks$ | 0.2 |
| $V \rightarrow people$ | 0.1 | $V \rightarrow people$ | 0.1 |
| $V \rightarrow fish$ | 0.6 | $V \rightarrow fish$ | 0.6 |

| Fish | people | fish | tanks |
|---|--|---|---|
| $S \rightarrow \text{fish} 0.006$ $N \rightarrow \text{fish} 0.2$ $V \rightarrow \text{fish} 0.6$ $\text{NP} \rightarrow \text{fish} 0.14$ $\text{VP} \rightarrow \text{fish} 0.06$ | $\mathbf{S \rightarrow NP VP}$ $0.9 * 0.14 * 0.01 = 0.00126$ $\mathbf{S \rightarrow V NP}$ $0.05 * 0.6 * 0.35 = 0.0105$ $\mathbf{VP \rightarrow V NP}$ $0.5 * 0.6 * 0.35$ $\mathbf{NP \rightarrow NP NP}$ $0.2 * 0.14 * 0.35$ | | |
| $S \rightarrow \text{NP VP} 0.9$ $S \rightarrow \text{V NP} 0.05$ $S \rightarrow \text{people} 0.001$ $S \rightarrow \text{fish} 0.006$ $\text{VP} \rightarrow \text{V NP} 0.5$ $\text{VP} \rightarrow \text{people} 0.01$ $\text{VP} \rightarrow \text{fish} 0.06$ $\text{NP} \rightarrow \text{NP NP} 0.2$ $\text{NP} \rightarrow \text{people} 0.35$ $\text{NP} \rightarrow \text{fish} 0.14$ $\text{NP} \rightarrow \text{tanks} 0.14$ $\text{N} \rightarrow \text{people} 0.5$ $\text{N} \rightarrow \text{fish} 0.2$ $\text{N} \rightarrow \text{tanks} 0.2$ $\text{V} \rightarrow \text{people} 0.1$ $\text{V} \rightarrow \text{fish} 0.6$ | $\mathbf{S \rightarrow NP VP}$ $0.9 * 0.35 * 0.06 = 0.0189$ $\mathbf{S \rightarrow V NP}$ $0.05 * 0.1 * 0.14 = 0.0007$ $\mathbf{VP \rightarrow V NP}$ $0.5 * 0.1 * 0.14$ $\mathbf{NP \rightarrow NP NP}$ $0.2 * 0.35 * 0.14$ | | |
| Exemplul 2 | | $S \rightarrow \text{fish} 0.006$ $N \rightarrow \text{fish} 0.2$ $V \rightarrow \text{fish} 0.6$ $\text{NP} \rightarrow \text{fish} 0.14$ $\text{VP} \rightarrow \text{fish} 0.06$ | $\mathbf{S \rightarrow V NP}$ $0.05 * 0.6 * 0.14 = 0.0042$ $\mathbf{VP \rightarrow V NP}$ $0.5 * 0.6 * 0.14$ $\mathbf{NP \rightarrow NP NP}$ $0.2 * 0.14 * 0.14$ |
| $N \rightarrow \text{tanks} 0.2$ $\text{NP} \rightarrow \text{tanks} 0.14$ | | | |

adaptat din Jurafsky and Martin, 2007

| Fish | people | fish | tanks |
|---|--|---|---|
| $S \rightarrow \text{fish} 0.006$ $N \rightarrow \text{fish} 0.2$ $V \rightarrow \text{fish} 0.6$ $\text{NP} \rightarrow \text{fish} 0.14$ $\text{VP} \rightarrow \text{fish} 0.06$ | $S \rightarrow \text{NP VP}$ $0.9 * 0.14 * 0.01 = 0.00126$ $S \rightarrow V \text{ NP}$ $0.05 * 0.6 * 0.35 = 0.0105$ $\text{VP} \rightarrow V \text{ NP} 0.5 * 0.6 * 0.35$ $\text{NP} \rightarrow \text{NP NP} 0.2 * 0.14 * 0.35$ | | |
| $S \rightarrow \text{NP VP} 0.9$ $S \rightarrow V \text{ NP} 0.05$ $S \rightarrow \text{people} 0.001$ $S \rightarrow \text{fish} 0.006$ $\text{VP} \rightarrow V \text{ NP} 0.5$ $\text{VP} \rightarrow \text{people} 0.01$ $\text{VP} \rightarrow \text{fish} 0.06$ $\text{NP} \rightarrow \text{NP NP} 0.2$ $\text{NP} \rightarrow \text{people} 0.35$ $\text{NP} \rightarrow \text{fish} 0.14$ $\text{NP} \rightarrow \text{tanks} 0.14$ $\text{N} \rightarrow \text{people} 0.5$ $\text{N} \rightarrow \text{fish} 0.2$ $\text{N} \rightarrow \text{tanks} 0.2$ $\text{V} \rightarrow \text{people} 0.1$ $\text{V} \rightarrow \text{fish} 0.6$ | $S \rightarrow \text{people} 0.001$ $N \rightarrow \text{people} 0.5$ $V \rightarrow \text{people} 0.1$ $\text{NP} \rightarrow \text{people} 0.35$ $\text{VP} \rightarrow \text{people} 0.01$ | $S \rightarrow \text{NP VP}$ $0.9 * 0.35 * 0.06 = 0.0189$ $S \rightarrow V \text{ NP}$ $0.05 * 0.1 * 0.14 = 0.0007$ $\text{VP} \rightarrow V \text{ NP} 0.5 * 0.1 * 0.14$ $\text{NP} \rightarrow \text{NP NP} 0.2 * 0.35 * 0.14$ | |
| | | $S \rightarrow \text{fish} 0.006$ $N \rightarrow \text{fish} 0.2$ $V \rightarrow \text{fish} 0.6$ $\text{NP} \rightarrow \text{fish} 0.14$ $\text{VP} \rightarrow \text{fish} 0.06$ | $S \rightarrow V \text{ NP}$ $0.05 * 0.6 * 0.14 = 0.0042$ $\text{VP} \rightarrow V \text{ NP}$ $0.5 * 0.6 * 0.14$ $\text{NP} \rightarrow \text{NP NP}$ $0.2 * 0.14 * 0.14$ |
| | | | $\text{N} \rightarrow \text{tanks} 0.2$ $\text{NP} \rightarrow \text{tanks} 0.14$ |

Exemplul 2

Chart incomplet

De completat de catre studenti

pt examen

De gasit cel mai probabil arbore de analiza

Analiza sintactica

- CYK pt GICP foloseste o **tabela de probabilitati P** de dimensiuni $M \times N \times N$ (M -nr neterm, N - nr cuv)
- $P(i,j,X)$ - contine probabilitatea maxima ca subsirul de la i la j sa fie generat de neterminalul X
- Intoarce matricea P cu aceste probabilitati calculate
- Foloseste si o matrice **Back(i,j,X)** care indica pentru subsirul de la i la j generat de $X \rightarrow Y Z$ unde s-a impartit subsirul a.i. sa fie generat de X

algoritm CYK(sir, gramatica) **intoarce** P , o tabela de probabilitati,
si T , cel mai probabil arbore

$N \leftarrow \text{Lungime}(\text{sir})$ /* sir = $a_1 a_2 \dots a_N$ */

$M \leftarrow \text{nr de neterminale in gramatica}$

$P \leftarrow \text{matrice de } N \times N \times M$, initial 0

Back \leftarrow matrice $N+1 \times N+1 \times M$

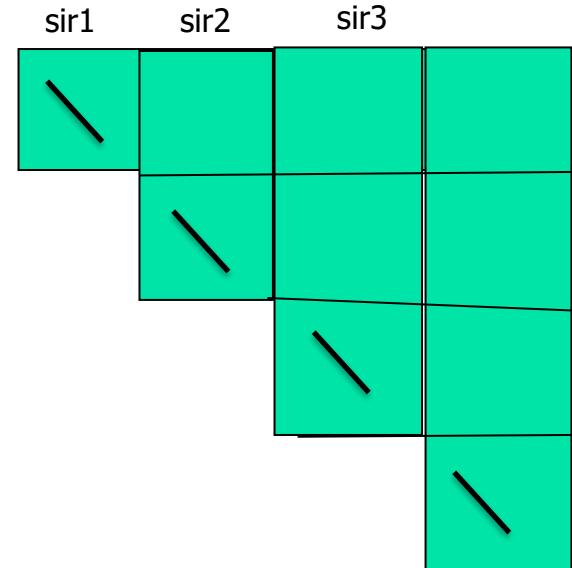
/* insereaza reguli lexicale pt fiecare cuvant */

pentru $i=0 .. N-1$ **repeta**

pentru fiecare regula de forma $(X \rightarrow \text{sir}_i, [p])$ **repeta**

$P[i, i+1, X] \leftarrow p$

(eventual si pointer catre regula)



algoritm CYK(sir, gramatica) **intoarce P**, o tabela de probabilitati, si **T**, cel mai probabil arbore

```
/* combina primul si al doilea neterminale  
din RHS a regulilor sintactice , incepand cu cele mai scurte */  
pentru span = 2 .. N repeta /* lungimea analizata */  
    pentru begin = 0 .. N-span repeta /* inceput analiza */  
        end  $\leftarrow$  begin + span  
        pentru split = begin+1 .. end-1 repeta /* partitie analizata */  
            pentru fiecare regula de forma ( $X \rightarrow Y Z$  [p]) repeta  
                daca p > P[begin, end, X] atunci  
                    P[begin, end, X]  $\leftarrow$   
                        P[begin, split, Y] x P[split, end, Z] x p  
                        (eventual si pointer catre regula)  
                    Back[begin,end, X]  $\leftarrow$  (split, Y,Z)  
intoarce P, construiesteArb(Back, P)  
sfarsit
```

Observatii

- Algoritmul se poate modifica pentru a trata si productii unare ($A \rightarrow B$)
- Algoritmul se poate modifica si pentru a trata productii vide
- Existenta productiilor cu 2 terminale este esentiala
- Exista si alti algoritmi de tip “chart parser”:
 - Earley – top down
 - Chart parsing – un fel de agenda-based

Analiza sintactica paritala

- Multe taskuri NLP nu necesita arbori de analiza completi (si deci de complexitate mare)
- Pentru aceste taskuri este suficient sa se realizeze o analiza paritala (*partial parse* or *shallow parse*)
- *Sistemele de extragere a informatiei* nu extrag toata informatie posibila din text, ci identifica si analizeaza acele segmente de text care contin informatii de interes
- *Sistemele de regasire a informatiei* pot indexa documentele pe baza unei subsmultimi de constituenti din text

5 Definite Clause Grammar (DCG)

- Utilizarea Logicii Predicatelor pentru reprezentarea gramaticilor
- Gramatici cu clauze definite (DCG)
- Fac parte din categoria gramaticilor semantice
- Fiecare regula din gramatica poate fi vazuta ca o regula din GIC

Definite Clause Grammar (DCG)

- Fiecare **categorie sintactica** se reprezinta printr-un **predicat cu un argument sir**
- NP(s) este adevarat daca s este NP
$$S \rightarrow NP \text{ VP } devine$$
$$NP(s1) \wedge VP(s2) \Rightarrow S(\text{append}(s1, s2))$$
- Parsing = inferenta logica
- Bottom-up parsing – forward chaining
- Top-down parsing – backward chaining
- Aceeasi gramatica poate fi utilizata atat pentru analiza cat si pentru generare

In BNF

$S \rightarrow NP\ VP$

In LP / DGC

$NP(s_1) \wedge VP(s_2) \Rightarrow S(\text{Append}(s_1, s_2))$

BNF

$\text{Noun} \rightarrow \text{ball} \mid \text{book}$

In LP / DGC

$(s = \text{"ball"} \vee s = \text{"book"}) \Rightarrow \text{Noun}(s)$

BNF, DCG, Prolog

| BNF | FOPL/DCG | PROLOG |
|---|--|---|
| $S \rightarrow NP\ VP$ $NP \rightarrow Noun$ $Noun \rightarrow stench$ $Noun \rightarrow wumpus$ $VP \rightarrow Verb$ $Verb \rightarrow smells$ $Verb \rightarrow kills$ | $NP(s1) \wedge VP(s2) \Rightarrow S(\text{append}(s1, s2))$ $Noun(s) \Rightarrow NP(s)$ $Verb(s) \Rightarrow VP(s)$ $(s = \text{"stench"} \vee s = \text{"wumpus"}) \Rightarrow Noun(s)$ $(v = \text{"smells"} \vee v = \text{"kills"}) \Rightarrow Verb(v)$ | <pre>sentence([S1, S2]) :- np(S1), vp(S2). np(S):- noun(S). vp(S):- verb(S). noun(stench). noun(wumpus). verb(smells). verb(kills). ?- sentence([wumpus, smells]). ?- sentence([S1, S2]).</pre> |

Imbogatire DCG

- Imbogatesc neterminale cu argumente suplimentare
- Argumentele suplimentare permit:
 - Verifica corectitudinea gramaticală
 - Atasează semantica
 - Adaugă expresii / funcții care se testează

Argument pt semantica

| DCG | FOPL | PROLOG |
|--|--|---------------|
| $S(sem) \rightarrow NP(sem1) VP(sem2)$ $\{compose(sem1, sem2, sem)\}$ | $NP(s1, sem1) \wedge VP(s2, sem2) \Rightarrow$ $S	append(s1, s2)),$ $compose(sem1, sem2, sem)$ | slide urmator |

semantica compozitionala

| | |
|---------------------|------------------------------------|
| The dog has legs. | (caine <i>parti</i> picioare) |
| The ball is yellow. | (minge <i>proprietate</i> galbena) |
| The ball is red. | (minge <i>proprietate</i> rosie) |
| The dog bites. | (caine <i>actiune</i> musca) |

```
sentence(S, Sem) :- np(S1, Sem1), vp(S2, Sem2), append(S1, S2, S),
                           Sem = [Sem1 | Sem2].
```

`np([S1, S2], Sem) :- article(S1), noun(S2, Sem).`

```
vp([S], Sem) :- verb(S, Sem1), Sem = [actiune, Sem1].
```

```
vp([S1, S2], Sem) :- verb(S1,_), adjective(S2, Sem1), Sem = [propriate, Sem1].
```

```
vp([S1, S2], Sem) :- verb(S1,_), noun(S2, Sem1), Sem = [parti, Sem1].
```

noun(dog,caine).

noun(ball,minge).

noun(legs,picioare).

verb(bytes,musca).

verb(is,este).

verb(has,are).

adjective(yellow,galbena).

adjective(red.rosie).

- S(sem) → NP(sem1) VP(sem2)
 - {compose(sem1, sem2, sem)}
- VP(sem) → verb(sem1)
 - {compose(**actiune**, sem1, sem)}
- VP(sem) → verb,adjective(sem1)
 - {compose(**proprietate**, sem1, sem)}
- VP(sem) → verb,noun(sem1)
 - {compose(**parti**, sem1, sem)}

| | |
|---------------------|------------------------------------|
| The dog has legs. | (caine <i>parti</i> picioare) |
| The ball is yellow. | (minge <i>proprietate</i> galbena) |
| The ball is red. | (mine <i>proprietate</i> rosie) |
| The dog bites. | (caine <i>actiune</i> musca) |

```
sentence(S, Sem) :- np(S1, Sem1), vp(S2, Sem2), append(S1, S2, S),
                           Sem = [Sem1 | Sem2].
```

```
np([S1, S2], Sem) :- article(S1), noun(S2, Sem).
```

`vp([S], Sem) :- verb(S, Sem1), Sem = [actiune, Sem1].`

```
vp([S1, S2], Sem) :- verb(S1,_), adjective(S2, Sem1), Sem = [propriete, Sem1].
```

```
vp([S1, S2], Sem) :- verb(S1,_), noun(S2, Sem1), Sem = [parti, Sem1].
```

noun(dog,caine).

noun(ball,ball).

noun(legs,picioare).

verb(bytes,musca).

verb(is,este).

verb(has,are).

adjective(yellow,galbena).

adjective(red,rosie).

```
?- sentence([the,ball,is,yellow],Sem).
```

Sem = [minge, proprietate, galbena]

Yes

```
?- sentence([the,dog,bytes],Sem).
```

Sem = [caine, actiune, musca]

Yes

```
?- sentence([is,dog,bytes],Sem).
```

No

?- sentence([the,dog,has,legs],Sem).

Verificare corectitudine gramaticală

- Cazuri
 - Subcategorii verbe: complementul pe care il poate accepta un verb
 - Acord subiect predicat
 - etc.
-
- Parametrizarea neterminalelor

Cazuri

Nominativ (subjective)

I take the bus

Eu iau autobuzul

You take the bus

Tu iezi autobuzul

He takes the bus

El ia autobuzul

Acuzativ (objective)

He gives me the book

Imi da cartea

S → NP(Subjective) VP

NP(case) → Pronoun (case) | Noun | Article Noun

//

I

VP → VP NP(Objective)

//

believe him

VP → VP PP

//

turn to the right

VP → VP Adjective

VP → Verb

PP → Preposition NP(Objective)

Pronoun(Subjective) → I | you | he | she

Pronoun(Objective) → me | you | him | her

sentence(S) :- np(S1,subjective), vp(S2), append(S1, S2, S).

np([S], Case) :- pronoun(S, Case).

np([S], _) :- noun(S).

np([S1, S2], _) :- article(S1), noun(S2).

pronoun(i, subjective).

pronoun(you, _).

pronoun(he, subjective).

pronoun(she, subjective).

pronoun(me, objective).

pronoun(him, objective).

pronoun(her, objective).

noun(ball).

noun(stick).

article(a).

article(the).

$S \rightarrow NP(\text{Subjective})\ VP$

$NP(\text{case}) \rightarrow \text{Pronoun (case)} \mid \text{Noun} \mid \text{Article Noun}$

$VP \rightarrow VP\ NP(\text{Objective})$

$VP \rightarrow VP\ PP$

$VP \rightarrow VP\ \text{Adjective}$

$VP \rightarrow \text{Verb}$

$PP \rightarrow \text{Preposition}\ NP(\text{Objective})$

$\text{Pronoun(Subjective)} \rightarrow I \mid \text{you} \mid \text{he} \mid \text{she}$

$\text{Pronoun(Objective)} \rightarrow \text{me} \mid \text{you} \mid \text{him} \mid \text{her}$

Subcategorii verbe

- Lista de subcategorii: ce complemente acceptă verbul; depinde de verb

$S \rightarrow NP(\text{Subjective}) \ VP(\text{subcat})$

$VP(\text{subcat}) \rightarrow \begin{cases} \{\text{subcat} = np\} \ VP(np) \ NP(\text{Objective}) \\ | \ \ \{\text{subcat} = adj\} \ VP(adj) \ \text{Adjective} \\ | \ \ \{\text{subcat} = pp\} \ VP(pp) \ PP \\ | \ \ \text{Verb} \end{cases}$

Subcategorii verbe

$\text{VP}(\text{subcat}) \rightarrow \begin{cases} \{\text{subcat} = \text{np}\} \text{ VP}(\text{np}) \text{ NP(Objective)} \\ | \quad \{\text{subcat} = \text{adj}\} \text{ VP}(\text{adj}) \text{ Adjective} \\ | \quad \{\text{subcat} = \text{pp}\} \text{ VP}(\text{pp}) \text{ PP} \\ | \quad \text{Verb} \end{cases}$

smell [NP] smell a wumpus

[Adjective] smell awfull

[PP] smell like a wumpus

is [Adjective] is smelly

[PP] is in box

[NP] is a pit

give [NP, PP] give the gold in box to me

[NP, NP] give me the gold

died [] died

Combinări cazuri cu subcategorii verbe

$S \rightarrow NP(\text{Subjective}) \ VP(\text{subcat})$

$NP(\text{case}) \rightarrow \text{Pronoun (case)} \mid \text{Noun} \mid \text{Article Noun}$

$\text{Pronoun}(\text{Subjective}) \rightarrow I \mid \text{you} \mid \text{he} \mid \text{she}$

$\text{Pronoun}(\text{Objective}) \rightarrow \text{me} \mid \text{you} \mid \text{him} \mid \text{her}$

$VP(\text{subcat}) \rightarrow \{ \text{subcat} = np \} \ VP(np) \ NP(\text{Objective})$

$\mid \{ \text{subcat} = adj \} \ VP(adj) \text{ Adjective}$

$\mid \{ \text{subcat} = pp \} \ VP(pp) \text{ PP}$

$\mid \text{Verb}$

$\mid VP(\text{subcat}) \text{ PP}$

$\mid VP(\text{subcat}) \text{ Adverb}$

$S \rightarrow NP(\text{Subjective}) \ VP(\text{subcat})$

$VP(\text{subcat}) \rightarrow \{\text{subcat} = np\} \ VP(np) \ NP(\text{Objective})$

- | {subcat = adj} VP(adj) Adjective
- | {subcat = pp} VP(pp) PP
- | Verb
- | **VP(subcat) PP**
- | VP(subcat) Adverb

Reprezentare in Prolog

`sentence(S) :- np(S1,subjective), vp(S2,Subcat), append(S1, S2, S).`

... pt NP

`vp([S],np) :- vp(S1,np), np(S2,objective), append(S1, S2, S).`

`vp([S],adj) :- vp(S1,adj), adjective(S2),append(S1, S2, S).`

`vp([S],pp) :- vp(S1,pp), pp(S2),append(S1, S2, S).`

`vp([S],_) :- verb(S).`

`vp([S],Subcat) :- vp(S1,Subcat), pp(S2),append(S1, S2, S).`

`vp([S],Subcat) :- vp(S1,Subcat), adverb(S2),append(S1, S2, S).`

Probleme datorita recursivitatii

Trebuie eliminata recursivitatea pentru a evita cicluri infinite

Solutie pentru eliminare recursivitate

```
vp([S],np) :- vp(S1,np), np(S2,objective), append(S1, S2, S).  
vp([S],Subcat) :- vp(S1,Subcat), pp(S2),append(S1, S2, S).
```

Se transforma in

```
vp([S], Subcat) :- Subcat = np, vp1(S1, np), np(S2, objective),  
append(S1, S2, S).  
vp([S],Subcat) :- vp1(S1, Subcat), pp(S2), append(S1, S2, S).
```

```
vp1([S],np) :- verb(S).
```

```
verb(give).
```

```
verb(make).
```

Gramatici semantice

- DCG au inspirat gramaticile semantice
- Gramaticile lingvistice (de ex GICP) utilizeaza neterminale sintactice (noun, verb, etc.)
- Gramaticile semnatice au pe post de neterminale categorii semantice, specifice unui domeniu

Top 10 artists in US for the last 3 weeks

limit

entity

geo

date

6. Analiza semantica

- Elementele de baza
 - Entitati / instante
 - Concepte / clase
 - Relatii – intre entitati si concepte
 - Predicate – verbe, roluri semantice

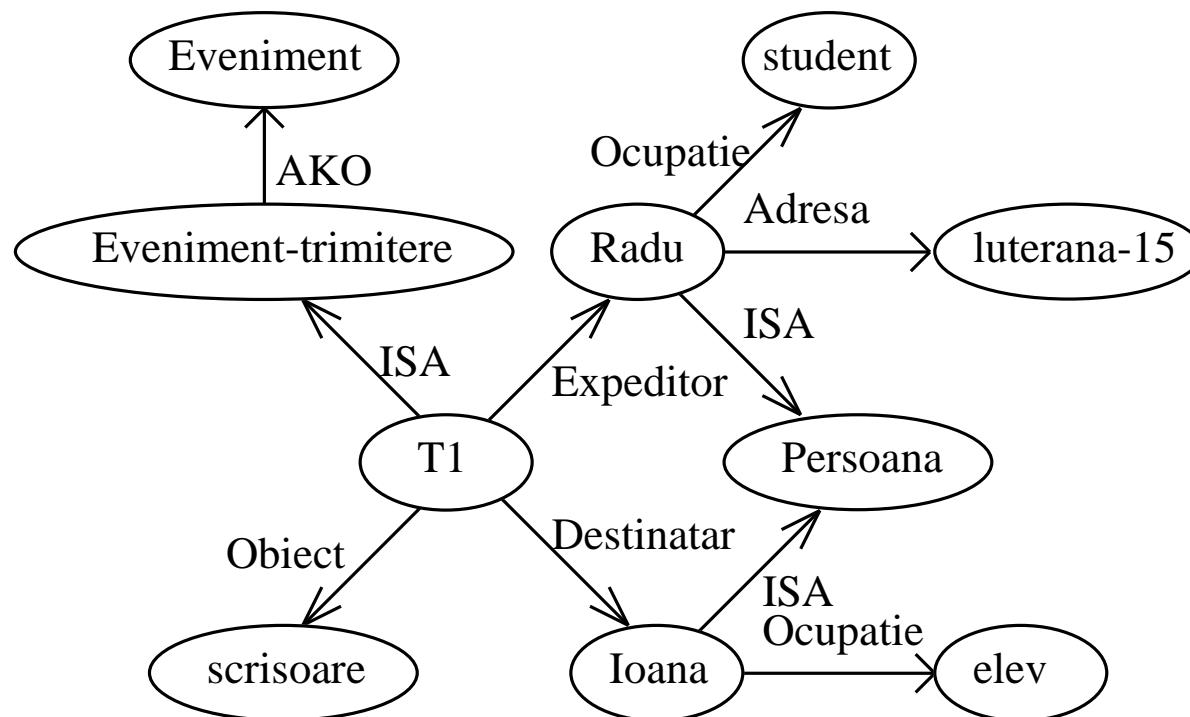
Analiza semantică - abordări

- Retele semantice
- Dependente conceptuale
- Gramatici de caz (Case Grammars)
- Grafuri conceptuale

Retele semantice

- Radu i-a trimis Ioanei o scrisoare.
- Radu este student.
- Ioana este eleva.
- Adresa lui Radu este Luterana, 15.

- Ocupatie (radu, student)
- Ocupatie (ioana, eleva)
- Trimite (radu, ioana, scrisoare)
- Adresa (radu, luterana - 15)



(b)

Retele semantice

■ **Mostenirea proprietatilor (atributelor)**

1) Mostenirii proprietatilor de la clasa la instantă:

Dacă un obiect O_1 este o particularizare (legat prin relația ISA) a unui obiect generic/ clasa O și clasa O are un atribut (proprietasă) A, atunci și instantă O_1 are atributul A.

2) Aplicarea mostenirii proprietatilor între o clasa și o superclasa, de-a lungul unei relații sau a unui lanț de relații AKO

Dacă o clasa C_1 este o subclasa a unei clase C (legată prin una sau mai multe relații AKO) și clasa C are proprietatea A, atunci clasa C_1 are de asemenea proprietatea (atributul) A.

7 Analiza pragmatică

- Interpretare pragmatică – utilizare și efect asupra ascultatorului
- Indexical – referă situația curentă, contextul (eu, acum, aici, atunci)
- Speech acts (comezi, asertiiuni)
- Ambiguități

Dezambiguizare

- Semantica lexicala
- **Elemente de ambiguitate lexicala**
 - Hiponimie – relatie intre un teren generic si instante ale termenului generic (culoare / rosu galben)
 - Omonimie – aceeasi ortografie dar sensuri diferite (broasca)
 - Polisemie – cuvinte cu sensuri diferite dar corelate (banca)
 - Sinonimie – forme lexicale diferite dar acelasi sens sau sensuri apropiate (vulnerable, firav, slab, nerezistent)
 - Antinomie – simetrie a intelelesului in fct de o axa, de ex proprietate, utilizare etc (agreabil / dezagreabil, adevarat/fals)

Dezambiguizare

- Sintactica – arbori diferiti de analiza
- Referentiala – referire la obiecte aneroioare
 - (Mihai i-a spus lui Tudor ca a luat examenul)
- Pragmatica – referire la loc, timp
- Ambiguitati intre semnificatia uzuala si figurativa
 - Metonimie (a numi un obiect prin intermediul unui termen desemnând un alt obiect, legat cu primul printr-o relatie logica)
 - Metafora (figura de stil prin care se trece de la sensul obișnuit al unui cuvânt la alt sens, prin intermediul unei comparații subînțelese)

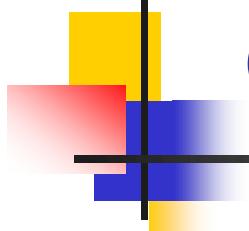


Inteligentă Artificială

Universitatea Politehnica Bucuresti
Anul universitar 2020-2021

Adina Magda Florea

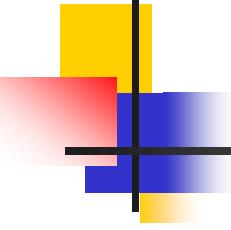




Curs nr. 12

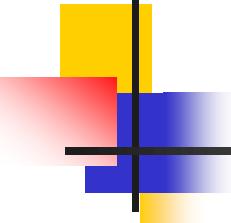
Prelucrarea limbajului natural

- a) Prelucrare LN pentru comunicare
(curs 11)
- b) Prelucrare LN pt achizitia cunostintelor



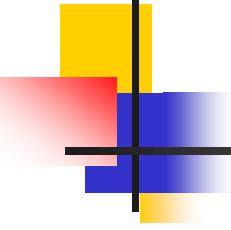
1 Modele ale limbajului

1. Modele de limbaj bazate pe structura gramaticală – (a)
 2. POS tagging (a și b)
 3. Word embedding
 4. Modele n-Gram
- 2, 3, 4 - IR, IE, sentiment analysis, etc.



1.1 POS tagging

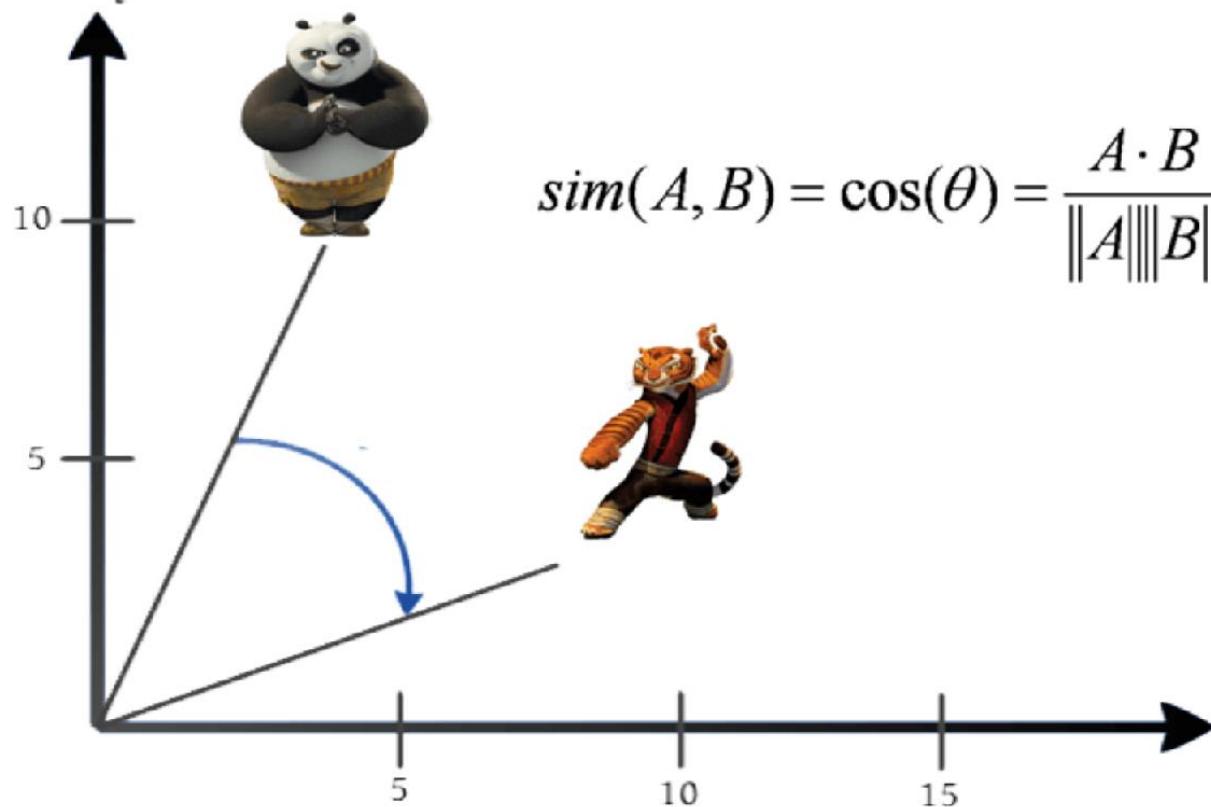
- **Part-of-speech tagging** – un cuvant sau un grup de cuvinte sunt etichetate cu o categorie gramaticală în funcție de semnificatie și de context (cuvinte adiacente, fraza, paragraf).
- Substantiv, adj, etc
- Gen, acord, forme infelxionate etc
- Engleză – de la 50 la 150 PoS
- Penn tag set
- Metode: HMM, Baum-Welch, SVM, NN

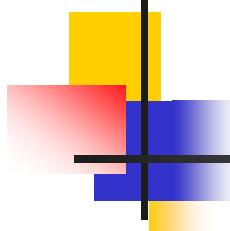


1.2 Word Embedding

- Reprezentare a vocabularului care surprinde contextul cuvintelor – similaritate semantica si sintactica, relatiile cu alte cuvinte
- Matrice de co-aparitie (co-occurrence) cu reducerea dimensionalitatii
- Modele probabiliste
- Word2Vec – tehnica pt a invata word embedding bazata pe RN
- Utilizate in IR, invatarea ontologiilor

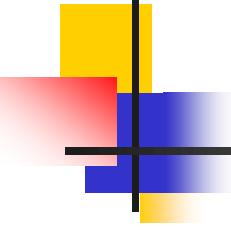
Cosine Similarity





Matrice de co-aparitie (Semantica distribuita)

- **Asocierea semantica** – co-aparitie in acelasi context (albina si miere)
- **Co-aparitie de ordinul I** (relatii sintagmatice – albina miere)
- **Co-aparitei de ordinul II** (relatii paradigmatice sau de similaritate – albina bondar)
- Se determina numarul de aparitii ale cuvintelor intr-o fereastra glisanta in text
- Se calculeaza similaritatea cosina

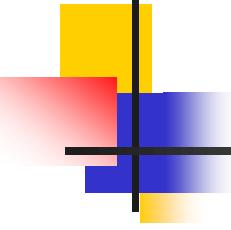


Matrice de co-aparitie

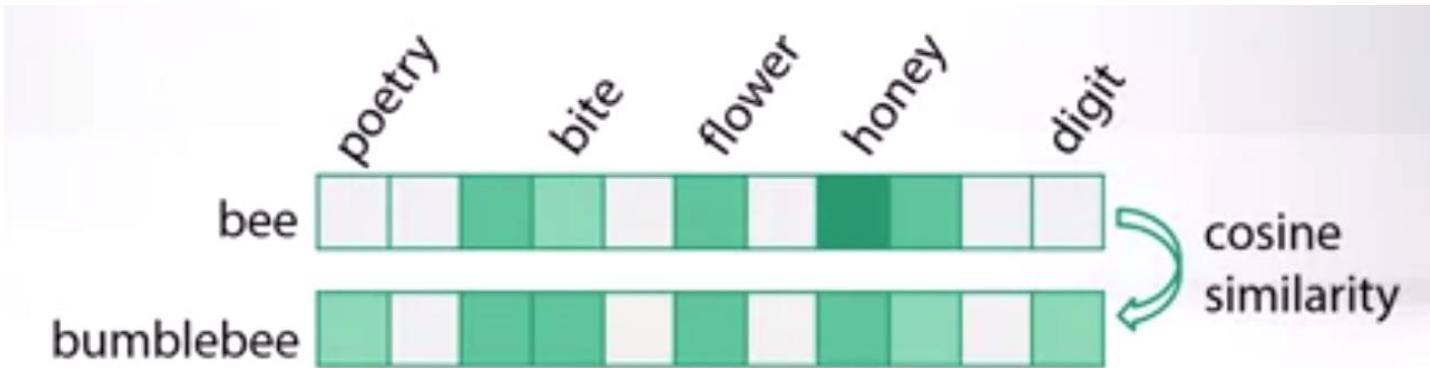


- Cum se calculeaza valorile din vector
- Pointwise Mutual Information

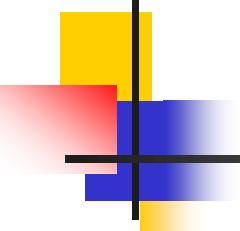
$$\text{PMI} = \log \left(p(u,v) / p(u) p(v) \right) = \log \left(n_{uv} / n_u n_v \right)$$



Matrice de co-aparitie

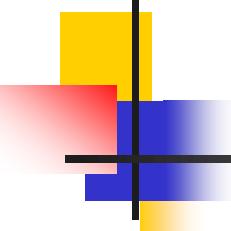


- Positive Pointwise Mutual Information
 $p\text{PMI} = \max(0, \text{PMI})$
- Calcul dificil



1.3 Modele N-gram

- **Model N-gram de caractere** – distributie de probabilitate peste secente de caractere
- $P(c_{1:N})$ – probabilitatea unei secente de N caractere, c_1 la c_N
 $P("the") = 0.27$ $P("zgq")=0.00000002$
- O secenta de simboluri de lungime n – **n-gram**
 - unigram, bigram, trigram
- Un model N-gram este definit ca un lant Markov de ordin N-1 (probabilitatea unui caracter depinde de caracterele precedente)



Modele N-gram de caractere

- **Trigram**

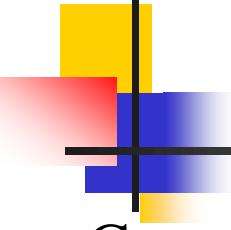
$$P(c_i | c_{1:i-1}) = P(c_i | c_{i-2:i-1})$$

$$P(c_{1:N}) = \prod_{i=1,N} P(c_i | c_{1:i-1}) = \prod_{i=1,N} P(c_i | c_{i-2:i-1})$$

Un model trigram a unui limbaj de 100 caractere

$$P(c_i | c_{i-2:i-1})$$

are 1 mil intrari



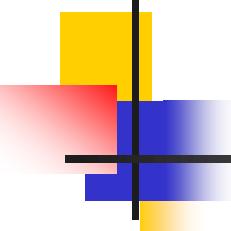
Modele N-gram de caractere

Ce putem face cu un astfel de model?

- **Identificarea limbajului** = fiind dat un text se determină în ce limbă este scris (99%)
- Construiește un model trigram caracter pentru fiecare limbaj candidat l
- $P(c_i|c_{i-2:i-1}, l)$; este nevoie de aprox 100 000 caractere pt fiecare limbaj

$$l^* = \operatorname{argmax}_l P(l|c_{1:N}) = \operatorname{argmax}_l P(l) * P(c_{1:N}|l) = \\ \operatorname{argmax}_l P(l) * \prod_{i=1, N} P(c_i|c_{i-2:i-1}, l)$$

- **Alte aplicații:**
 - verificare ortografie, clasificare texte în funcție de tipuri, identificarea numelor proprii,...



Modele N-gram de caractere

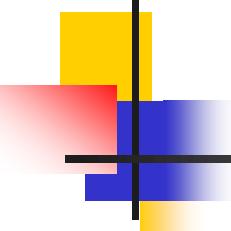
Omogenizarea modelelor

- *Problema:* pt secvente de caractere comune, cam orice corpus va da o estimare buna ($P(" th")=0.15$)
- Dar $P(" ht")=0$ [http?](http://)
- *Solutie*
- Calculam N-gram si pentru secvente cu $P=0$ sau f mica
- Calculam N-1-gram + interpolare

$$P(c_i|c_{i-2:i-1}, l) = \alpha_1 P(c_i|c_{i-2:i-1}, l) + \alpha_2 P(c_i|c_{i-1}, l) + \alpha_3 P(c_i)$$

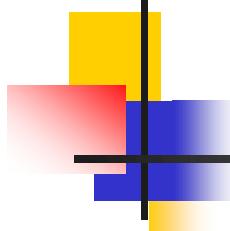
cu $\alpha_1 + \alpha_2 + \alpha_3 = 1$

Evaluarea modelelor – prin validare incrusisata



Modele N-gram de cuvinte

- In acest caz vocabularul este mult mai mare
- Daca max 100 car in cele mai multe limbaje, sute de mii, milioane de cuvinte
- Cuvinte noi
- Putem adauga un cuvant <NEC> in vocabular (sau mai multe)
- Bigramele si trigramele sunt mai bune decat unigramele



Modele N-gram de cuvinte - ex

Fie acest mic corpus

<s> I am Sam </s>

<s> Sam and I </s>

<s> I do not like green eggs and ham </s>

Calculul probabilitatilor unor bigrame pt acesta

$$P(I|<s>) = 2/3 = 0.67$$

$$P(Sam|<s>) = 1/3 = 0.33$$

$$P(am|I) = 1/3 = 0.33$$

$$P(</s>|Sam) = 1/2 = 0.5$$

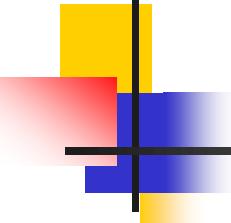
$$P(and|Sam) = 1/2 = 0.5$$

$$P(do|I) = 1/3 = 0.33$$

Daca avem probabilitatile tuturor bigramelor putem calcula

$$P(<s> Sam and I </s>) =$$

$$P(<Sam|<s>)*P(and|Sam)*P(I|and)*P(</s>|I)$$



2 Clasificarea textelor

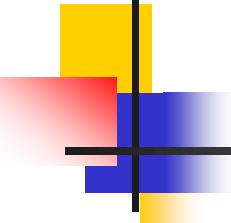
- Clasificare sau apartenență la o clasa
- Identificare limbaj, clasificare tip text, analiza stării induse, detectie spam

Detectie spam

| | |
|----------------|------|
| Not-spam (Ham) | Spam |
|----------------|------|

| | |
|------------|------------|
| m1, m2,... | n1, n2,... |
|------------|------------|

- "for cheap" "you can buy" – n-gram de cuvinte
- "yo,u d-eserve" – n-gram de caractere



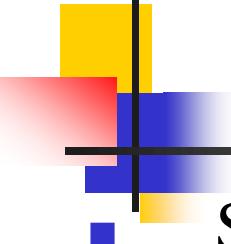
Detectie spam

- Calculez $P(\text{Mesaj}|\text{Spam})$ $P(\text{Mesaj}|\text{Ham})$
- Clasific mesaj nou

$$\underset{\mathcal{C} \in \{\text{Spam}, \text{Ham}\}}{\operatorname{argmax}} P(\mathcal{C}|\text{Mesaj}) = \underset{\mathcal{C} \in \{\text{Spam}, \text{Ham}\}}{\operatorname{argmax}} P(\text{Mesaj}|\mathcal{C}) * P(\mathcal{C})$$

unde $P(\mathcal{C})$ este estimat prin numararea nr de mesaje din Spam si Ham

- Se reprezinta mesajul ca o multime de caracteristici $(\text{car}_i, \text{val}_i)$ si se poate aplica un algoritm de clasificare (invatare)
- Caracteristici – cuvinte din vocabular
- Valori – nr de aparitii in mesaj
- Alg posibili: K-nearest-neigh, SVM, AD, Bayes naiv

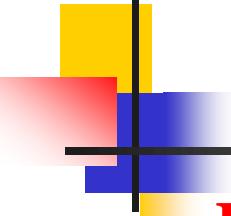


3 Regasirea informatiei

- Scop: Gasirea documentelor (de obicei text nestructurat) care sunt relevante pentru o cerere utilizator, dintr-o colectie de documente

Un sistem de IR (Information Retrieval) poate fi caracterizat de:

- **Un corpus de documente** – paragrafe, pagini, texte pe mai multe pagini
- **Interrogarea (query) in limbajul de interrogare** – lista cuvinte, cuvinte adiacente, op logici, op nelogici (near)
- **Multimea rezultat** – multimea de documente relevante pentru query
- **Prezentarea rezultatelor** – lista ordonata, grafic, etc.

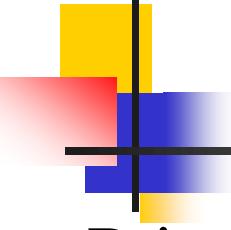


Evaluarea rezultatului

- **Precizie (precision)** – procentul de documente relevante din multimea obtinuta
- **Relevanta (recall)** – procentul de documente relevante din colectie care sunt in setul rezultat
- 100 documente cu 1 query pt care obtinem o multime de 40

| | In multime | Nu in multime |
|------------|------------|---------------|
| Relevant | 30 | 20 |
| Nerelevant | 10 | 40 |

- **Precision** = $30/(30+10) = 0.75$
- **Recall** = $30/(30+20) = 0.6$
- Recall este mai greu de calculat
- Se pot combina $2PR/(P+R)$



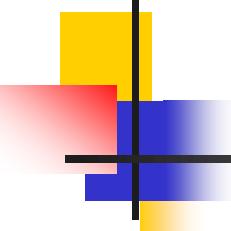
Regasirea Booleană

Primele sisteme IR – Boolean keyword model

- Fiecare cuvant din text tratat ca un flag
- Limbajul de interogare – expresii logice peste cuvinte
- Ce piese ale lui Shakespeare contin cuvintele ***Brutus AND Caesar but NOT Calpurnia?***

| | Antony and Cleopatra | Julius Caesar | The Tempest | Hamlet | Othello | Macbeth |
|------------------|----------------------|---------------|-------------|--------|---------|---------|
| Antony | 1 | 1 | 0 | 0 | 0 | 1 |
| Brutus | 1 | 1 | 0 | 1 | 0 | 0 |
| Caesar | 1 | 1 | 0 | 1 | 1 | 1 |
| Calpurnia | 0 | 1 | 0 | 0 | 0 | 0 |
| Cleopatra | 1 | 0 | 0 | 0 | 0 | 0 |
| mercy | 1 | 0 | 1 | 1 | 1 | 1 |
| worser | 1 | 0 | 1 | 1 | 1 | 0 |

1 daca piesa continut cuvantul, 0 altfel



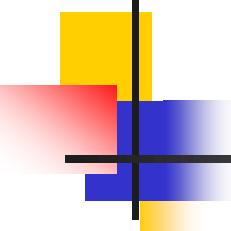
Regasirea Booleană

Vectori booleni (0/1)

Pentru a raspunde la query: se iau vectorii pt Brutus, Caesar si Calpurina (complementat)

- 110100 *AND*
- 110111 *AND*
- 101111 =
- **100100**

| | Antony and Cleopatra | Julius Caesar | The Tempest | Hamlet | Othello | Macbeth |
|------------------|----------------------|---------------|-------------|--------|---------|---------|
| Antony | 1 | 1 | 0 | 0 | 0 | 1 |
| Brutus | 1 | 1 | 0 | 1 | 0 | 0 |
| Caesar | 1 | 1 | 0 | 1 | 1 | 1 |
| Calpurnia | 0 | 1 | 0 | 0 | 0 | 0 |
| Cleopatra | 1 | 0 | 0 | 0 | 0 | 0 |
| mercy | 1 | 0 | 1 | 1 | 1 | 1 |
| worser | 1 | 0 | 1 | 1 | 1 | 0 |



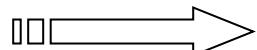
Regasirea Booleană

Pt 1 milion de documente cu aprox 1000 cuvinte fiecare,
rezulta o matrice f mare

O reprezentare mai buna: index inversat

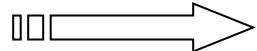
Pt fiecare termen se memoreaza lista documentelor care
contin t

Brutus



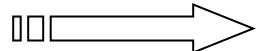
| | | | | | | | |
|---|---|---|----|----|----|-----|-----|
| 1 | 2 | 4 | 11 | 31 | 45 | 173 | 174 |
|---|---|---|----|----|----|-----|-----|

Caesar



| | | | | | | | |
|---|---|---|---|---|----|----|-----|
| 1 | 2 | 4 | 5 | 6 | 16 | 57 | 132 |
|---|---|---|---|---|----|----|-----|

Calpurnia



| | | | | | | | |
|---|----|----|-----|--|--|--|--|
| 2 | 31 | 54 | 101 | | | | |
|---|----|----|-----|--|--|--|--|

Dictionar

Postings

Constructia indexului inversat

Documente de indexat



Friends, Romans, countrymen.

⋮

Tokenizer

Tokens

Friends

Romans

Countrymen

Linguistic modules

Tokens modificate

friend

roman

countryman

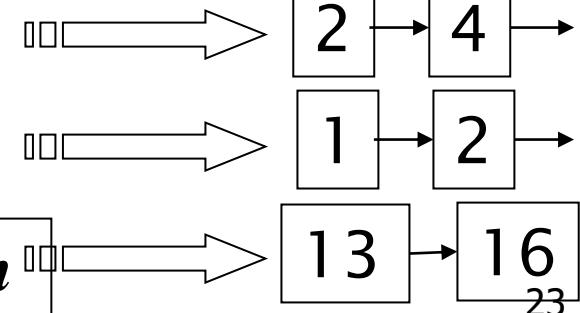
Indexer

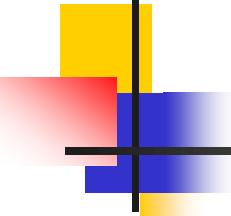
Index inversat

friend

roman

countryman





Etape initiale in prelucrarea textului

- Tokenization
 - Determina token (analiza lexicala)
 - ***John's***
- Normalizare
 - Mapeaza textul si termenul din query la aceeasi forma
 - ***U.S.A.*** si ***USA***
- Stemming
 - Forme diferite pt o aceeasi radacina sa se potriveasca
 - ***authorize, authorization***
- Eliminarea cuvintelor comune (stop words)
 - ***the, a, to, of***

Secventa tokens

| Term | docID |
|-----------|-------|
| I | 1 |
| did | 1 |
| enact | 1 |
| julius | 1 |
| caesar | 1 |
| I | 1 |
| was | 1 |
| killed | 1 |
| i' | 1 |
| the | 1 |
| capitol | 1 |
| brutus | 1 |
| killed | 1 |
| me | 1 |
| so | 2 |
| let | 2 |
| it | 2 |
| be | 2 |
| with | 2 |
| caesar | 2 |
| the | 2 |
| noble | 2 |
| brutus | 2 |
| hath | 2 |
| told | 2 |
| you | 2 |
| caesar | 2 |
| was | 2 |
| ambitious | 2 |
| | |
| | |
| | |
| | |
| | |
| | |
| | |

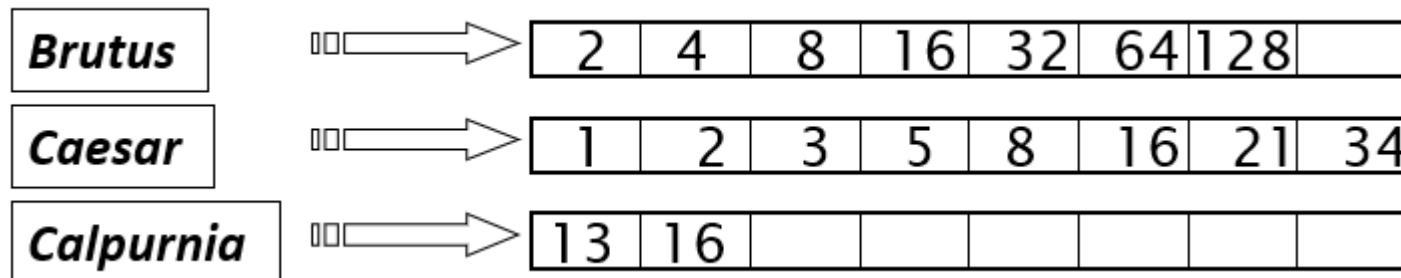
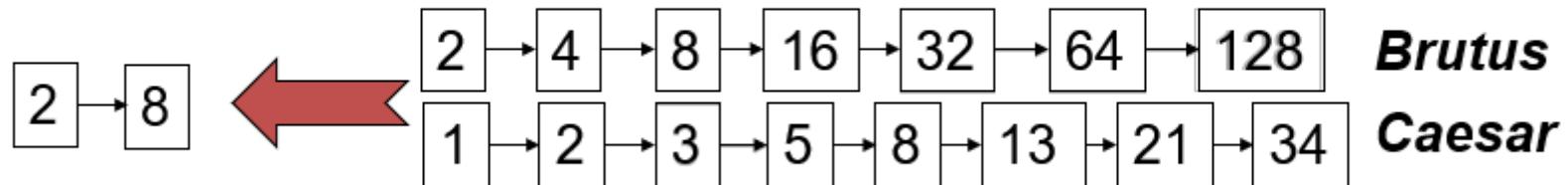
Secventa tokens sortata

| Term | docID |
|-----------|-------|
| ambitious | 2 |
| be | 2 |
| brutus | 1 |
| brutus | 2 |
| capitol | 1 |
| caesar | 1 |
| caesar | 2 |
| caesar | 2 |
| did | 1 |
| enact | 1 |
| hath | 1 |
| I | 1 |
| I | 1 |
| i' | 1 |
| it | 2 |
| julius | 1 |
| killed | 1 |
| killed | 1 |
| let | 2 |
| me | 1 |
| noble | 2 |
| so | 2 |
| the | 1 |
| the | 2 |
| told | 2 |
| you | 2 |
| was | 1 |
| was | 2 |
| with | 2 |
| | |
| | |
| | |
| | |
| | |
| | |
| | |

Dictionar si postings lists

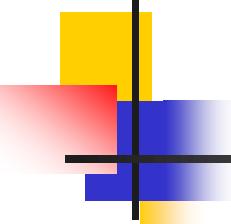
| term | doc. | freq. | → | postings lists |
|-----------|------|-------|---|----------------|
| ambitious | 1 | 1 | → | 2 |
| be | 1 | 1 | → | 2 |
| brutus | 2 | 1 | → | 1 → 2 |
| capitol | 1 | 1 | → | 1 |
| caesar | 2 | 1 | → | 1 → 2 |
| did | 1 | 1 | → | 1 |
| enact | 1 | 1 | → | 1 |
| hath | 1 | 1 | → | 2 |
| i | 1 | 1 | → | 1 |
| i' | 1 | 1 | → | 1 |
| it | 1 | 1 | → | 2 |
| julius | 1 | 1 | → | 1 |
| killed | 1 | 1 | → | 1 |
| let | 1 | 1 | → | 2 |
| me | 1 | 1 | → | 1 |
| noble | 2 | 1 | → | 2 |
| so | 2 | 1 | → | 2 |
| the | 1 | 1 | → | 1 → 2 |
| the | 2 | 1 | → | 2 |
| told | 1 | 1 | → | 2 |
| you | 1 | 1 | → | 2 |
| was | 2 | 1 | → | 1 → 2 |
| with | 1 | 1 | → | 2 |

Prelucrare integrogari



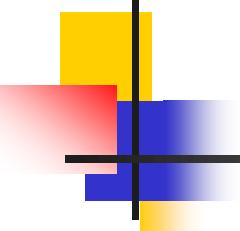
Brutus AND Caesar but NOT Calpurnia

(*Calpurnia AND Brutus*) AND *Caesar*



Regasirea pe baza de rang

- În regasirea Booleană documentele fie se potrivesc fie nu
- Utilizatorul trebuie să introducă un query Boolean (uneori dificil)
- Numărul de documente gasite este mare sau 0
- **Regasirea bazată pe rang (ranked retrieval systems)** – se produce un set ordonat de documente relevante pentru query
- Nu mai folosesc un limbaj de interogare ci interogare în limbaj natural
- Ideea de bază – **frecvența termenilor (Term Frequency)** = numărul de aparitii ale unui termen în document
- Modele statistice bazate pe contoare de cuvinte

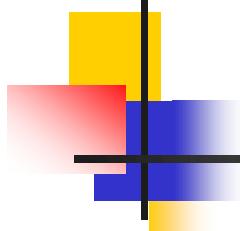


Regasirea pe baza de rang

| | Antony and Cleopatra | Julius Caesar | The Tempest | Hamlet | Othello | Macbeth |
|-----------|----------------------|---------------|-------------|--------|---------|---------|
| Antony | 157 | 73 | 0 | 0 | 0 | 0 |
| Brutus | 4 | 157 | 0 | 1 | 0 | 0 |
| Caesar | 232 | 227 | 0 | 2 | 1 | 1 |
| Calpurnia | 0 | 10 | 0 | 0 | 0 | 0 |
| Cleopatra | 57 | 0 | 0 | 0 | 0 | 0 |
| mercy | 2 | 0 | 3 | 5 | 5 | 1 |
| worser | 2 | 0 | 1 | 1 | 1 | 0 |

Modelul bag of words – reprezentarea cu vectori nu consideră ordinea cuvintelor

John is quicker than Mary și *Mary is quicker than John* au același vector

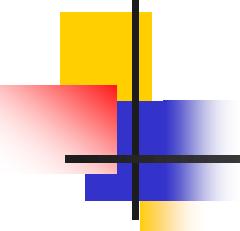


Term frequency (tf)

- **Frecventa termenilor (Term Frequency)**
- $f_{t,d}$ = numarul de aparitii ale termenului **t** in documentul **d**
- Relevanta unui document nu creste proportional cu frecventa termenilor (un document cu 10 aparitii a lui t este mai relevant decat un document cu 1 aparitie, dar nu de 10 ori mai relevant)
- Se defineste varianta logaritmica a **term frequency**

$$tf_{t,d} = \begin{cases} 1 + \log_{10} f_{f,d} & \text{daca } f_{f,d} > 0 \\ 0 & \text{in caz contrar} \end{cases}$$

0 -> 0, 1->1, 2->1.3, 10->2, 1000 ->4



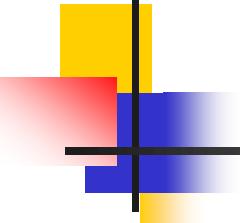
Term frequency (tf)

- **Frecventa augmentata** – impartita la frecventa cuvantului care apare cel mai des in document

$$tf_{t,d} = 0.5 + 0.5 * f_{t,d} / \max(f_{t',d} : t' \in d)$$

- Scorul pt termenul **t** pt query **q** si document **d**

$$\text{Scor} = \sum_{t \in q \cap d} tf_{t,d}$$



Document frequency si idf

- Aparitia in document a unor cuvinte rare este mai relevanta decat cea a cuvintelor comune
- **Frecventa in documente** = numarul de documente in care apare t
- **d_{ft} – document frequency pt t**
- Este o masura inversa a cat de informativ este documentul si $d_{ft} \leq N$, cu $N=|D|$, nr total de documente din corpusul D
- **Inverse document frequency** pt t

$$idf_{t,D} = \log_{10} (N/df_t)$$

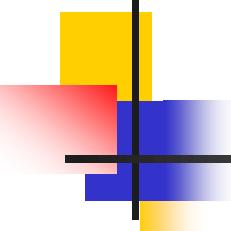
Inverse document frequency

$N = 1 \text{ milion}$

$$\mathbf{idf}_{t,D} = \log_{10} (N/df_t)$$

| term | df _t | idf _t |
|-----------|-----------------|------------------|
| calpurnia | 1 | 6 |
| animal | 100 | 4 |
| sunday | 1,000 | 3 |
| fly | 10,000 | 2 |
| under | 100,000 | 1 |
| the | 1,000,000 | 0 |

idf - *cata informatie ofera un cuvant scade ponderea cuvintelor care apar frecvent in documente si creste ponderea celor care apar rar*



tf si idf

$$tf_{t,d} = 1 + \log_{10} t_{f,d}$$

$$idf_{t,D} = \log_{10} (N/df_t)$$

$$tfidf_{t,d,D} = tf_{t,d} * idf_{t,D}$$

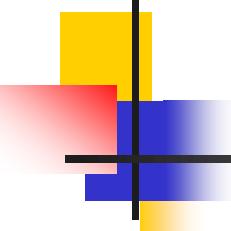
tfidf_{t,d,D}

- Creste odata cu numarul de aparitii in document
- Creste odata cu raritatea cuvintelor in colectie
- O valoare mare a **tfidf_{t,d,D}** se obtine pentru o frecventa mare a cuvantului in document si o valoare mica a frecventei cuvantului in colectia de documente (se incearca eliminarea cuvintelor comune)

Matrice de ponderi

| Antony | Antony and Cleopatra | Julius Caesar | The Tempest | Hamlet | Othello | Macbeth |
|-----------|----------------------|---------------|-------------|--------|---------|---------|
| | 5.25 | 3.18 | 0 | 0 | 0 | 0.35 |
| Brutus | 1.21 | 6.1 | 0 | 1 | 0 | 0 |
| Caesar | 8.59 | 2.54 | 0 | 1.51 | 0.25 | 0 |
| Calpurnia | 0 | 1.54 | 0 | 0 | 0 | 0 |
| Cleopatra | 2.85 | 0 | 0 | 0 | 0 | 0 |
| mercy | 1.51 | 0 | 1.9 | 0.12 | 5.25 | 0.88 |
| worser | 1.37 | 0 | 0.11 | 4.15 | 0.25 | 1.95 |

Fiecare document este reprezentat printr-un vector cu valori reale – ponderile **tfidf**



tf si idf

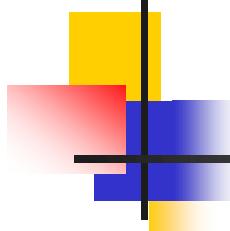
Ponderea (weight) a unui termen este

$$w_{t,d} = \log(1 + tf_{t,d}) \times \log_{10}(N / df_t)$$

Cea mai cunoscuta formula in IR

Creste cu nr de aparitii in document

Creste cu raritatea cuvintelor in colectie



Scorul unui document pt un query

$$\text{Score}(q, d) = \sum_{t \in q \cap d} \text{tf.idf}_{t,d}$$

Exemplu sistemul Lucene

$TF(q_i, d_j)$ pt N documente – nr de aparitii q_i in documentul d_j

$DF(q_i)$ – **Document frequency counts** – nr de documente care contin cuvantul q_i

Fiind dat un document d_j si un query cu cuvintele $q_{1:N}$ avem

$$BM25(d_j, q_{1:N}) = \sum_{i=1}^N IDF(q_i) * \frac{TF(q_i, d_j) * (k + 1)}{TF(q_i, d_j) + k(1 - b + b \frac{|d_j|}{L})}$$

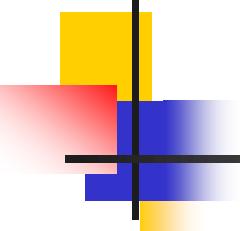
$|d_j|$ - nr cuvinte din documentul d_j

$L = \sum_i |d_i| / N$ – lungimea medie a documentelor din corpus

k, b – determinati prin validare incrusata, valori tipice:

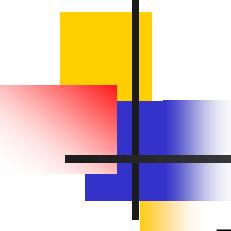
$$k=2.0, b=0.75$$

$$IDF(q_i) = \log \frac{N - DF(q_i) + 0.5}{DF(q_i) + 0.5}$$



Regasirea informatiei

- Dificil de aplicat tfidf fiecarui document din corpus
- **Hit list** = index creat anterior care refera pentru fiecare cuvant din vocabular documentele ce contin acel cuvant
- Pt un query, se face intersectia intre *hit list* si cuvintele din query si se face cautarea numai pe aceasta intersectie
- tfidf – model care trateaza cuvintele ca fiind independente
- **Imbunatatiri**
 - Corelatii
 - Cuvinte derivate, omonime



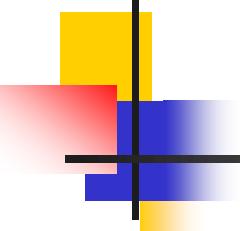
Regasirea informatiei

- Page Rank (Brin and Larry Page 1998 Google)
- In loc sa caute documente cu masuri IR cat mai mari, dau prioritate documentelor de calitate
- Paginile cu multe *in-links* au scor mare
- Page Rank – o distributie de probabilitate – persoana navigheaza pe Web sa ajunga la o anumita pagina

$$PR(p) = \sum_{in_i \in B_p} \frac{PR(in_i)}{C(in_i)}$$

- PR(p) – rangul paginii p
- B_p – multimea paginilor care au link la pagina p
- in_i – paginile care au link la p (paginile din B_p)
- C(in_i) – nr de *out-links* in pagina in_i

Ex: A, B->A,C, C-> A, D-> A,B,C PR(A) = PR(B)/2 + PR(C)/1 + PR(D)/3

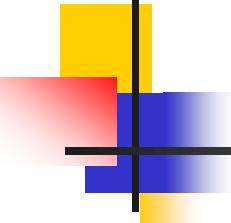


Regasirea informatiei

$$PR(p) = \frac{1-d}{N} + d \sum_{in_i \in Bp} \frac{PR(in_i)}{C(in_i)}$$

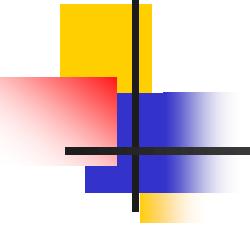
- PR(p) – rangul paginii p
- Bp – multimea paginilor care au link la pagina *p*
- in_i – paginile care au link la p (paginile din Bp)
- C(in_i) – nr de *out-links* in pagina in_i
- N – nr total de pagini in corpus
- d – damping factor – probabilitatea ca sa ramana pe aceeasi pagina
- Calculat iterativ – se incepe cu paginile cu
 $PR(p,0)=1/N$ si itereaza actualizand rangurile pana la convergenta

$$PR(p,t+1) = \frac{1-d}{N} + d \sum_i \frac{PR(in_i, t)}{C(in_i)}$$



4 Extragerea informatiei

- **Sisteme IE scop:**
- Localizeaza si extrage parti relevante dintr-un text
- Achizitia cunostintelor prin analiza unui text cu focalizare pe aparitia unei clase particulare de obiecte si a relatiilor dintre aceste obiecte
- Produce o reprezentare structurata
 - Relatii (intr-o bd)
 - Baza de cunostinte, ontologii



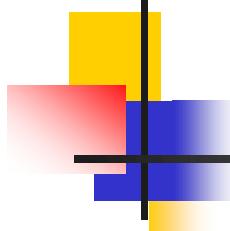
Extragerea informatiei

- Exemple tipice
 - extragerea din pagini web a adreselor (cu campuri strada, nr, etc.)
 - meteo – temp, vat, precipitatii, etc.

Capitala Iranului este orasul Teheran

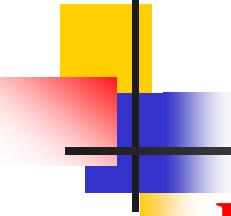
[capitala\(Iran, teheran\)](#)

- De ce este dificil?
- Ex: Pretul unui produs (un interval was \$100 now \$120, offers in the high 700s..)



Named Entity Recognition (NER)

- Gasirea si clasificarea entitatilor nume, de ex nume persoane, organizatii, locatii (orase) sau expresii numerice de ex timp, data, bani, procent etc.
- **Viorel Salvador Caragea**, fost șef al **Inspectoratului Județean de Poliție Gorj**, a demisionat din **USR**, după ce cu numai o zi înainte filiala județeană a **USR** i-a acceptat adeziunea. (sursa Digi24)
- Mazare fina congelata 450gr **5,99** ron

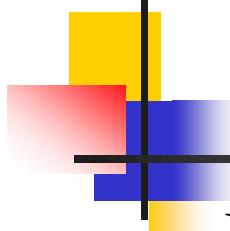


Evaluarea rezultatului

- **Precizie (precision)** – procentul de entitati selectate care sunt corecte
- **Relevanta (recall)** – procentul de entitati corecte care sunt selectate
- 100 documente din care selectez entitate nume companie XX

| | Corecte | Incorecte |
|--------------|---------|-----------|
| Selectate | 30 | 20 |
| Neselectatet | 10 | 40 |

- **Precision** = $30/(30+20) = 0.6$
- **Recall** = $30/(30+10) = 0.75$



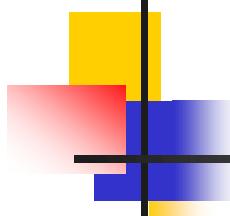
Evaluarea rezultatului

- De obicei o masura combinata (medie armonica)

$$F = \frac{1}{\alpha \frac{1}{P} + (1-\alpha) \frac{1}{R}} = \frac{(\beta^2 + 1)PR}{\beta^2 P + R}$$

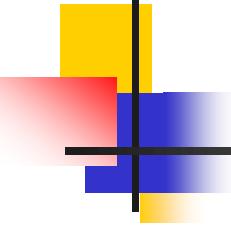
- Mai frecvent se utilizeaza cu $\beta=1$

$$F = 2PR/(P+R)$$



Abordari pentru NER

- Expresii regulate (automate finite)
- ML cu modele secentiale
 - Naive Bayes
 - Discriminative: modele Maxent
- Modele probabiliste
 - HMM
 - CRFs
- Utilizate si in combinatie cu POS tagging, analiza sintactica (NP, VP, PP) sau categorii semantice (din WordNet de ex)

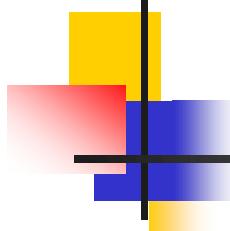


NER cu expresii regulate

■ Atomate finite

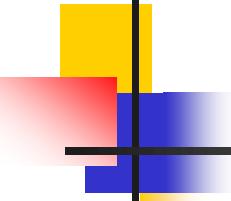
Templates

- Extrag informatii relevante unui obiect – valorile unor atribute predefinite
- Se defineste un template (pattern) pentru fiecare atribut de extras
- Template-ul este definit de un automat finit (expresii regulate)
- Template – prefix regex, target regex, postfix regex
- price prefix regex **[€][0-9]*[.][0-9][0-9]** postfix regex



NER cu expresii regulate

- Daca template-ul se potriveste 1 data – extrage target regex
- Daca nu se potriveste de loc – atribut lipsa
- Daca se potriveste de mai multe ori – prioritate, mai multe versiuni de template (prefix regex de ex)
- Intereseaza in special **Recall**



NER cu reguli de productie

- Ce persoana are ce pozitie intr-o organizatie

[person], [office] of [org]

Ian Flex, leader of Geen movement

[org] (named, appointed etc.) [person] Prep [office]

NATO appointed Mircea Geoana as Deputy Secretary General

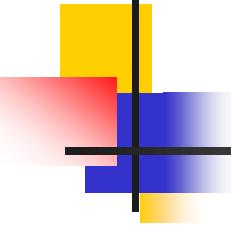
- Determinarea locatiei unei organizatii

[org] in [loc]

NATO headquarters in Brussels

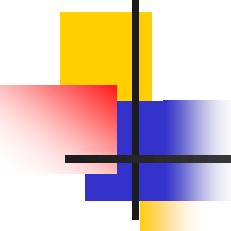
[org] [loc] (division, branch, headquarters, etc.)

Google branch in Vancouver



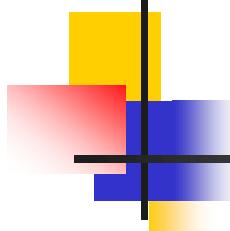
ML cu modele secentiale

- Antrenare
 - Colecțează set de documente reprezentative
 - Etichetează fiecare token cu clasa entității
 - Extrage attribute semnificative pentru clasa
 - Antrenează modelul pt a prezice clase
- Testare
 - Colecțează setul de documente de test
 - Rulează recunoașterea
 - Evaluatează rezultatele (R, P)



ML cu modele secentiale

- De ex "Named Entity Recognition and Classification with Scikit-Learn"
- Corpus adnotat cu IOB si POS tags
- IOB tags = Inside, Outside, Beginning
 - I- prefix - inaintea unui tag indica ca tagul este in interiorul unui chunk
 - B- prefix - inaintea unui tag indica ca un tag este la inceputul unui chunk.
 - O tag - indica ca un token nu apartine unui chunk (outside).



ML cu modele secentiale

■ Atribute ale entitatilor

geo = Geographical Entity

org = Organization

per = Person

gpe = Geopolitical Entity

tim = Time indicator

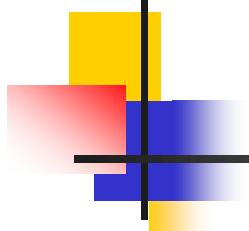
art = Artifact

eve = Event

nat = Natural Phenomenon

| A Sentence # | A Word | A POS | A Tag |
|--------------|---------------|-------|-------|
| Sentence: 1 | Thousands | NNS | 0 |
| | of | IN | 0 |
| | demonstrators | NNS | 0 |
| | have | VBP | 0 |
| | marched | VBN | 0 |
| | through | IN | 0 |
| | London | NNP | B-geo |
| | to | TO | 0 |
| | protest | VB | 0 |
| | the | DT | 0 |
| | war | NN | 0 |
| | in | IN | 0 |
| | Iraq | NNP | B-geo |
| | and | CC | 0 |
| | demand | VB | 0 |
| | the | DT | 0 |
| | withdrawal | NN | 0 |
| | of | IN | 0 |
| | British | JJ | B-gpe |
| | troops | NNS | 0 |
| | from | IN | 0 |
| | that | DT | 0 |
| | country | NN | 0 |
| | . | . | 0 |
| Sentence: 2 | Families | NNS | 0 |
| | of | IN | 0 |
| | soldiers | NNS | 0 |

- Anumite sisteme combina Information retrieval cu information extraction



Slide-uri pct 3 bazate pe

CS 276 / LING 286: Information Retrieval and Web Search



Universitatea
Politehnica
Bucuresti
Anul universitar
2020-2021

Adina Magda
Florea

Inteligentă Artificială



1

Continut curs

- Introducere în IA.
- Strategii de căutare
- Strategii în jocuri
- Reprezentarea cunoștințelor prin logica simbolică
- Reprezentarea cunoștințelor pe bază de reguli
- Planificare automată
- Raționament incert: probabilități, modelul euristic, rețele bayesiene
- Agenți și sisteme multi-agent
- Prelucrarea limbajului natural

Materiale curs

- A. Florea. Slide-uri curs
- S. Russell, P. Norvig. *Artificial Intelligence: A Modern Approach*, Prentice Hall, 2010 (editia a 3a),
<http://aima.cs.berkeley.edu/>
- D. Poole, A. Mackworth. *Artificial Intelligence: Foundations of Computational Agents*, Cambridge University Press, 2010 – complete book online
<http://artint.info/html/ArtInt.html>

Resurse AI on-line

- <http://aima.cs.berkeley.edu/index.html>
- Laboratorul de Inteligenta Artificiala si Sisteme Multi-agent (AI-MAS)
<http://aimas.cs.pub.ro/links>
- Asociația Română de Inteligență Artificială
<http://www.aria-romania.org/>
- Tools for learning AI
<http://www.aispace.org/index.shtml>
- Direcțiile de cercetare din IA:
<http://aitopics.org/>

Cursuri AI on-line

MIT Open Courseware

<http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-034-artificial-intelligence-fall-2010/>

Udacity

<https://www.udacity.com/course/intro-to-artificial-intelligence--cs271>

Diferite subiecte de IA – Coursera

<https://www.coursera.org/courses?categories=cs-ai>

Cerințe pentru promovare

- Participare la laborator: minimum 7 sedințe de laborator
- Minim 50% din punctajul de parcurs și minim 50% din punctajul de la examenul final
- Activitate și teme de laborator
- Rezolvarea temelor de casă
- Parcurgerea materialelor obligatorii
- Examen final

Notare

- Examen final: 40%
- Laborator 25%
- Teme de casă 25%
- Teste curs 10%

Curs nr. 1

Introducere în IA

Intrebări cheie

Este posibilă simularea comportamentului intelligent pe calculator?

Care este criteriul pe baza căruia se apreciază inteligența unui program?

La ce nivel se încearcă modelarea comportamentului intelligent?

Care sunt reprezentările și tehniciile utilizate în rezolvarea problemelor de inteligență artificială?

1. Ce este inteligența artificială?

- Alan Turing - “Computing Machinery and Intelligence”, 1950
- Loebner prize, 1990 -...
- IA abordare simbolică
- IA abordare non-simbolică

Definiții IA

- Inteligența artificială este studiul facultăților mentale pe baza modelelor computaționale.
- Un program intelligent este un program care manifestă o comportare similară cu aceea a omului când este confruntat cu o problemă similară. *Nu este necesar ca programul sa rezolve sau să încerce sa rezolve problema în același mod în care ar rezolva-o oamenii.*
- Abilitatea de a executa sarcini și de a rezolva probleme care sunt executate și/sau rezolvate de inteligență naturală, în particular de inteligență umană

Definiții IA

- Definiția Inteligenței Artificiale a Grupului de experți la nivel înalt pe probleme de IA al UE, 2019
- „*Sistemele de inteligență artificială (IA) sunt sisteme software (și, eventual, hardware) proiectate de oameni care au un obiectiv complex, acționează în dimensiunea fizică sau digitală, percepând mediul prin intermediul preluării datelor, prin interpretarea datelor structurate sau nestructurate colectate, prin raționare cu privire la cunoștințe sau prin prelucrarea informațiilor obținute din aceste date și prin deciderea celor mai bune acțiuni care trebuie întreprinse pentru a realiza obiectivul dat. Sistemele IA pot să utilizeze reguli simbolice sau să învețe un model numeric și își pot adapta comportamentul analizând modul în care mediul este afectat de acțiunile lor anterioare.*”

2. Caracteristicile problemelor IA

- Generale
- Dinamica modelului
- Dificile de rezolvat (**complexitatea calcului**)
- Cunoștințe versus date
- Utilizarea cunoștințelor euristice
- Utilizarea cunoștințelor incerte
- Necesită raționament, inferențe
- Comportament autonom
- Adaptare/învățare

Inferențe

- Inferență
- Regulă de inferență
 - Consistentă (sound) vs. inconsistentă
 - Completă vs. incompletă
- Strategie de inferență (control al inferentelor)
 - Consistentă vs. inconsistentă
 - Completă vs. incompletă

Exemple de reguli de inferență

A

$A \rightarrow B$

B

Inferențe deductive

Modus ponens

frumos(mircea)

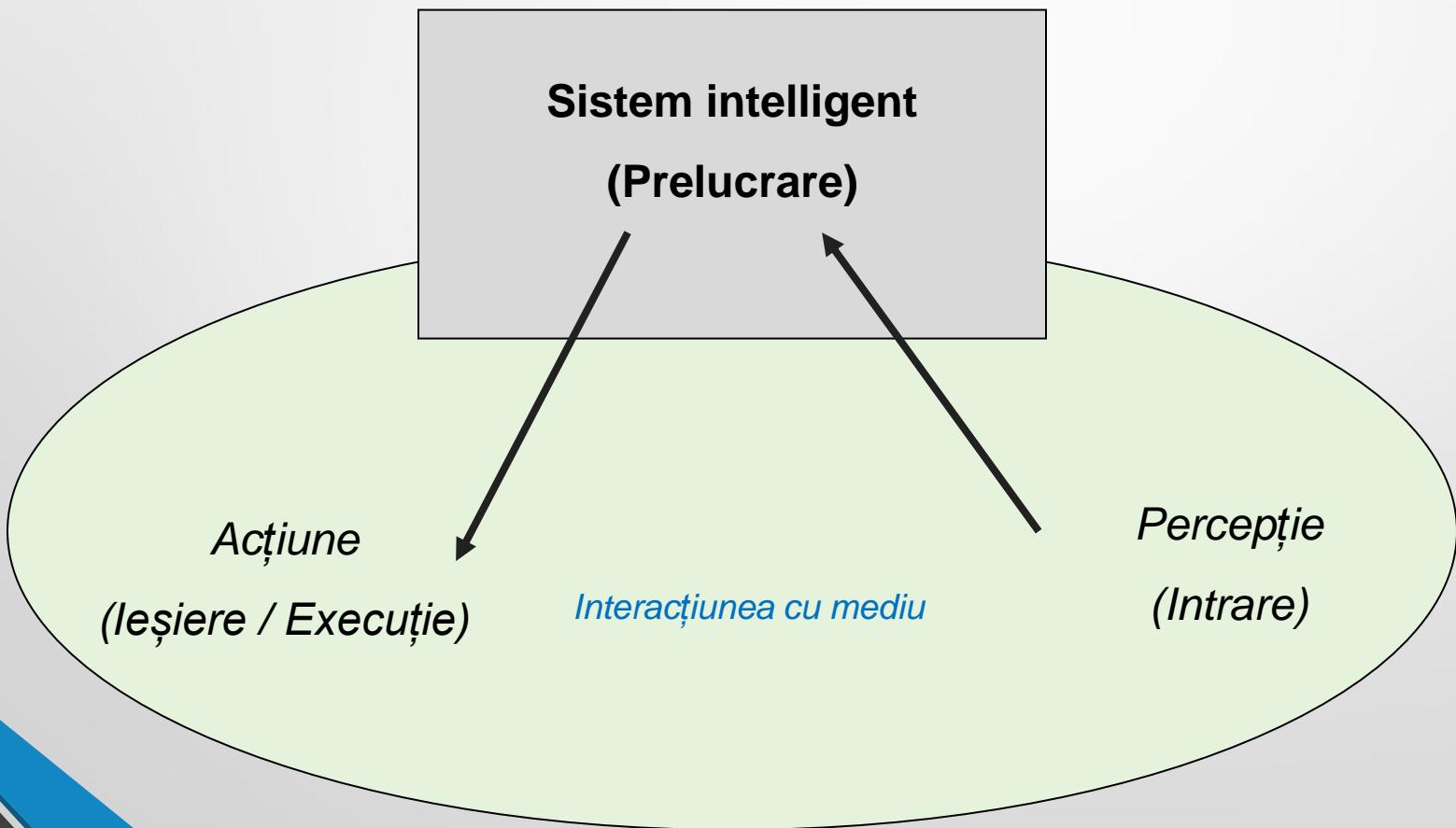
frumos(dan)

$\forall x$ frumos(x)

Inferențe nedeductive

Inferență inductivă

3. Structura unui sistem IA



- Sistemele AI pot folosi reguli simbolice sau pot învăța un model numeric și, de asemenea, își pot adapta comportamentul analizând modul în care mediul este afectat de acțiunile lor anterioare
- **Intrarea** sistemului poate fi reprezentată de videocamere, microfoane, text introdus de la tastatură sau senzori care detectează cantități fizice precum temperatura, viteza etc.
- Datele colectate pot fi structurate, adică date organizate conform unui model predefinit (de exemplu baze de date relaționale) sau date nestructurate, date care nu au o structură organizată, de exemplu, o imagine, o zonă de text sau un semnal sonor.

- Partea de **prelucrare** realizează analiza datelor și luarea deciziilor (raționament).
- Se presupune că un sistem de IA acționează, respectiv decide rațional, în sensul că ia decizia corectă pentru atingerea scopului sau rezolvarea problemei.
- Termenul de decizie nu implică neapărat totala autonomie a sistemului. Majoritatea sistemelor actuale de IA sunt semi-autonome, oferind o recomandare operatorului uman, care va fi factorul de decizie final.
- **Ieșirea** este răspunsul sistemului pe baza deciziilor luate
- Elementele de execuție (actuatorare) nu trebuie neapărat să fie fizice, pot fi de asemenea componente software, rezultate prezentate utilizatorului uman etc.

- Capacitățile sistemelor de IA pot fi grupate în două grupuri principale:
 - **capacitatea de raționament a sistemului** - este realizată prin modele care includ reprezentarea cunoștințelor și utilizarea acestora în luarea deciziilor, planificare, căutare și optimizare.
 - **capacitatea de a învăța** – este realizată prin modele cum ar fi rețele neurale, învățare profundă (*deep learning*), sisteme de suport vectoriale (SVM), metode de grupare (*clustering*), algoritmi genetici și altele.
- Aceste tehnici permit unui sistem de IA să învețe cum să rezolve probleme care nu pot fi specificate cu exactitate sau pentru care metoda de rezolvare nu poate fi descrisă prin reguli de raționament simbolic.

- Un aspect important la ora actuală este acela al cuplării abordărilor de IA cu robotica.
- Robotica poate fi definită ca „**IA în acțiune în lumea fizică**” (numită și IA încorporată).
- Un robot este o mașină fizică care trebuie să facă față dinamicii, incertitudinilor și complexității lumii fizice.
- Percepția, raționamentul, acțiunea, învățarea, precum și capacitatele de interacțiune cu alte sisteme sunt, de obicei, integrate în arhitectura de control a sistemului robotizat.

4. Scurt istoric

- Conferința de la Dartmouth College din 1956 - primii patru mari inițiatori ai domeniului: John McCarthy, Marvin Minsky, Alen Newell si Herbert Simon.
- 1956 - 1957 A. Newell, J. Shaw si H. Simon - primul program de demonstrare automată a teoremelor, "The Logic Theorist."
- 1959 Samuel Checkers program
- Începând din 1960 apar primele programe de inteligență artificială.

Istoric

- 1965 J. A. Robinson – rezoluția
- 1965 – DENDRAL - J. Lederberg si E. Feigenbaum. - sistem expert capabil să sintetizeze structura moleculelor organice pe baza formulelor chimice și a spectogramelor de masă
- 1959 - Limbajul Lisp (LISt Processing) - John McCarthy (Dartmouth College)
- 1972 - Limbajul Prolog (PROgrammation et LOGique) - Alain Colmerauer (universitatea Marseille-Aix)
- 1983 - Smalltalk - Goldberg, Robson

Istoric

- Anii '70 – importanța cunoștințelor
- **Sisteme bazate pe cunoștințe**
- **Ingineria cunoștințelor**
- Sistemul MYCIN - Buchanan, Shortliffe - sistem expert pentru diagnosticarea infecțiilor bacteriene ale sângeului, Stanford University - '74-'75
- Sisteme expert
- Sisteme cadru pentru dezvoltarea sistemelor expert

Istoric

- Anii '80-'90 – dezamăgire
- Anii '90-'00 – relansare IA
- **IA distribuită**
- **Agenți inteligenți**
- **Sisteme multi-agent**
- Din ce în ce mai multe programe, componente – inteligente
- **Anii 2010 - 2020**

IA omniprezentă

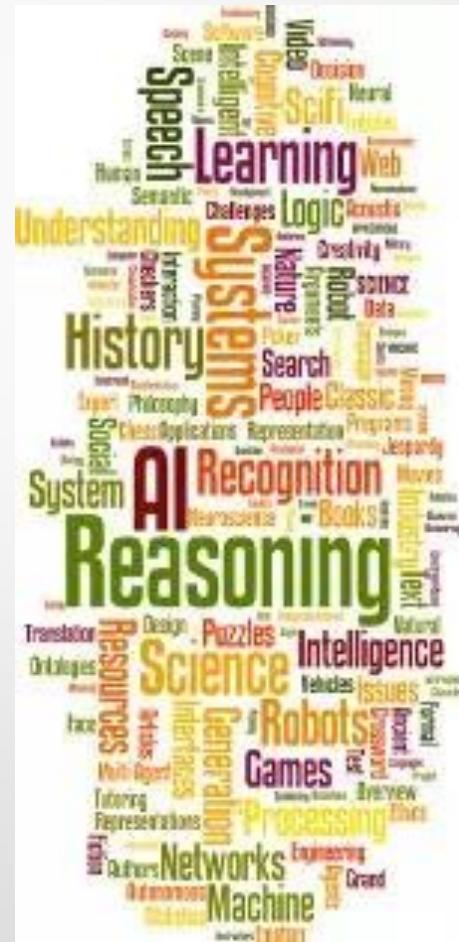
Istoric

Actual / 2020

- Combinare
- Strong AI
- Machine learning
- Deep learning:
 - deep neural networks
 - deep belief networks
 - recurrent neural networks
 - deep reinforcement learning

5. Domeniile IA

- Învățare automată
 - Vedere computerizată
 - Înțelegerea limbajului natural
 - Sinteza automată a vorbirii
 - Reprezentarea cunoștințelor
 - Raționament automat
 - Căutarea soluțiilor
 - Jocuri
 - Agenți inteligenți și sisteme multi-agent
 - Demonstrarea automată a teoremelor
 - Expertiză: inginerie, medicină, analiză financiară, sisteme de suport a deciziei, predicție, etc.
 - Robotică



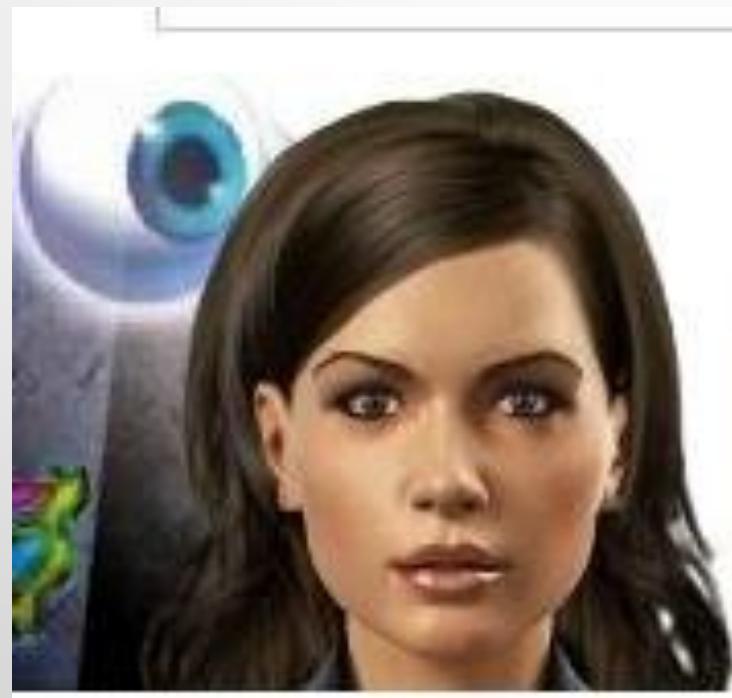
IA la ora actuală

- Omniprezentă:
 - comunicării
 - conducere procese
 - conducere vehicule
 - investiții financiare
 - supraveghere și operații de salvare
 - medicină
 - aplicații web,

6. Aplicații

Alice agent (bot)

[A. L. I. C. E. The Artificial Linguistic Internet Computer Entity](#)



- În 1997 super-computerul *Deep Blue* l-a învins pe campionul de șah Gary Kasparov.
- Actual, calculatoarele pot juca șah mai bine decât orice campion



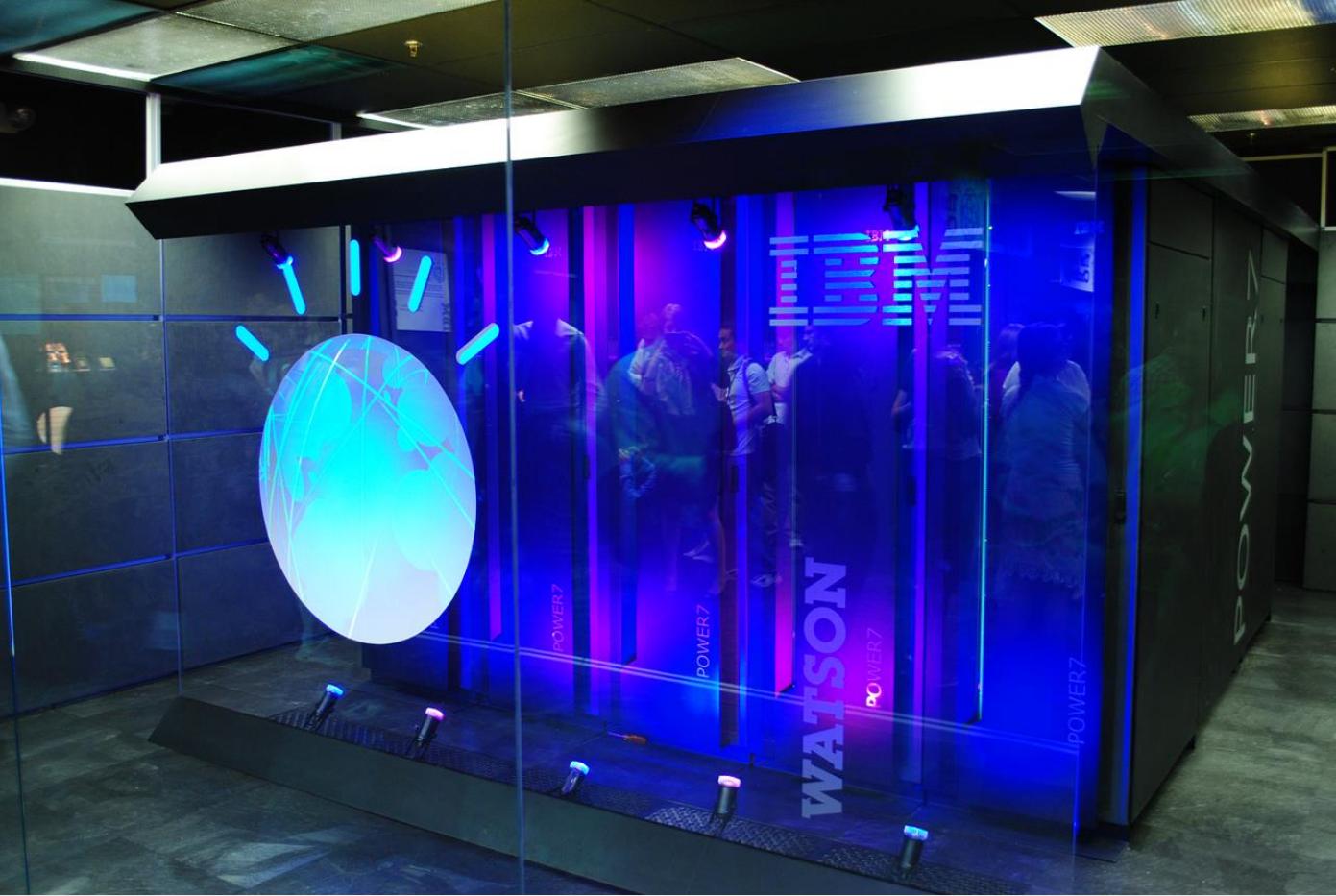


MoGo învinge pe Myungwan Kim,
august 2008



Sixth Sense Project

- Proiectul Sixth Sense – conceput de Pranav Mistry, cercetător la Massachusetts Institute of Technology's Media Lab.



- **Watson**
- Proiectul IBM DeepQA
- Întrebări în limbaj natural pentru quiz show *Jeopardy*
- În 2011, Watson a concurat cu Brad Rutter and Ken Jennings și a obținut locul I - \$1 million.
- 200 milioane de pagini, 4 terabytes de disc

- **\$400** : With much "Gravity", this young fellow of Trinity became the Lucasian Professor of Mathematics in 1669

Isaac Newton

- **\$400** : Some hedgehogs enter periods of torpor; the Western European species spends the winter in this dormant condition

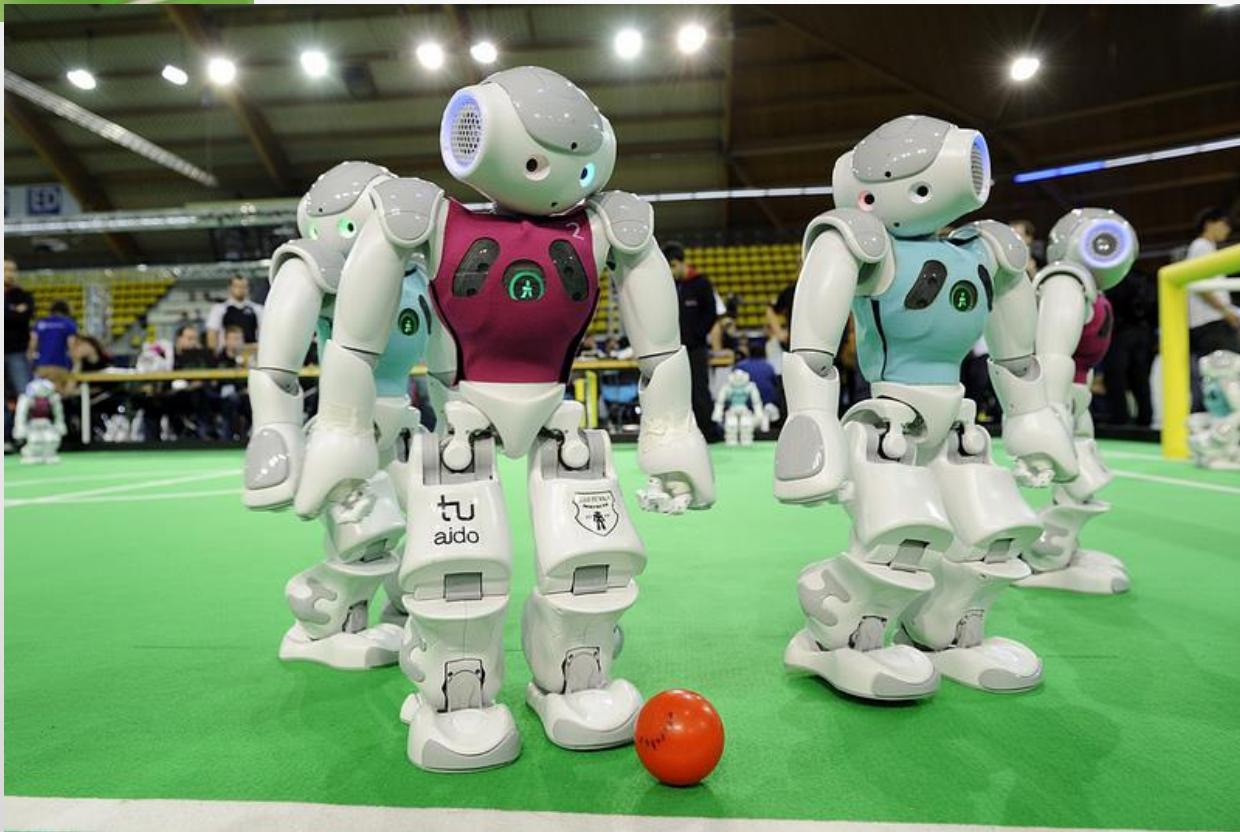
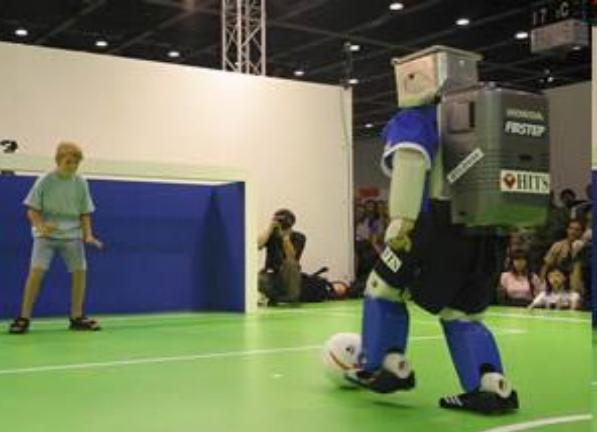
Hibernation

- **\$400** : This title gal, "Children at your feet, wonder how you manage to make ends meet"

Lady Madonna

- **\$800** : The first flight takes place at Kitty Hawk & baseball's first World Series is played

1900's



Competiția Robocup

Mașina autonomă –Google Chauffeur



Deep learning

2011 Andrew Ng - Google's Google Brain -
Rețea neurală antrenata cu algoritmi deep learning

– recunoașterea conceptelor cum ar fi pisica pe baza YouTube videos

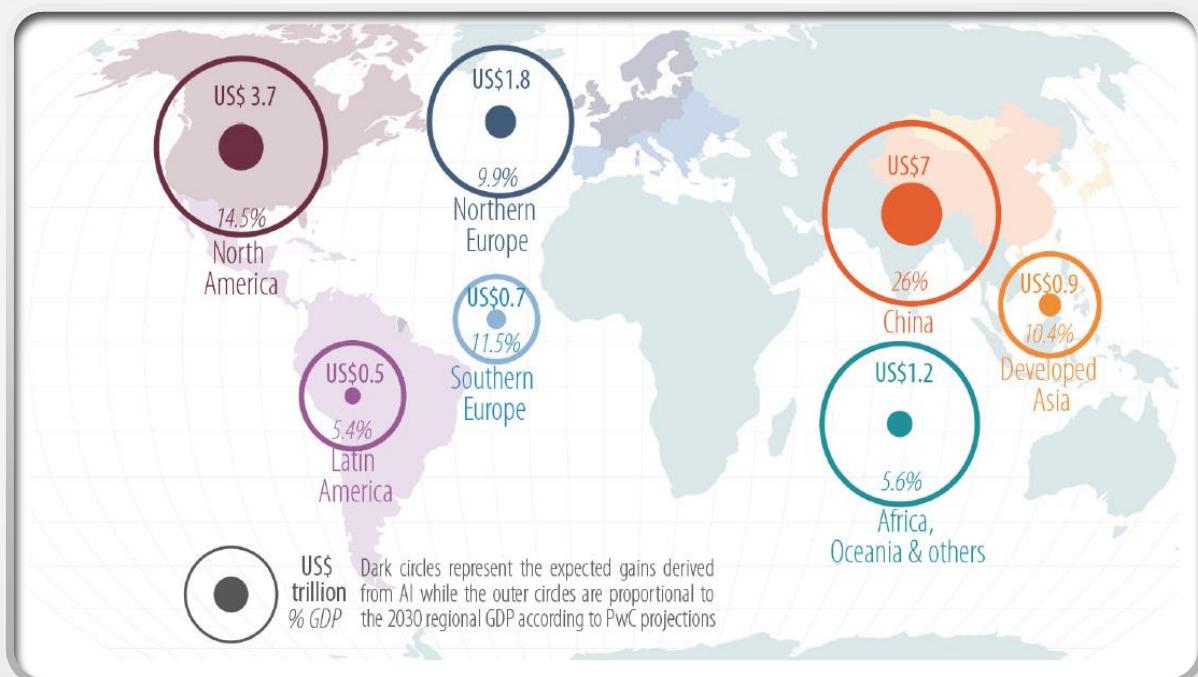
Facebook Yann LeCun - director of AI Research – deep learning pentru identificarea fețelor și a obiectelor în cele 350 milioane de fotografii incarcate pe Facebook

7. Impactul Inteligenței Artificiale

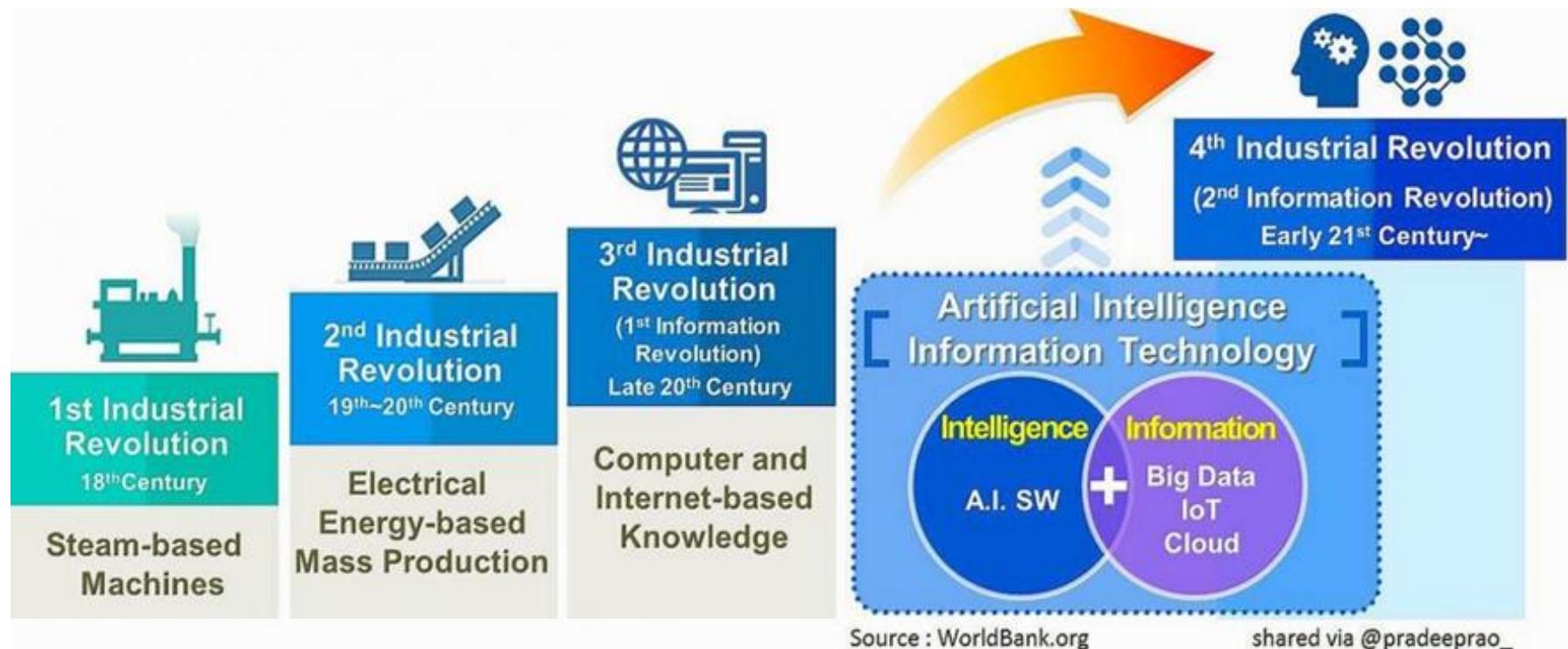
- Majoritatea studiilor a subliniat faptul că Inteligența Artificială va avea un ***impact economic și social*** semnificativ la nivel global

Impactul Inteligenței Artificiale

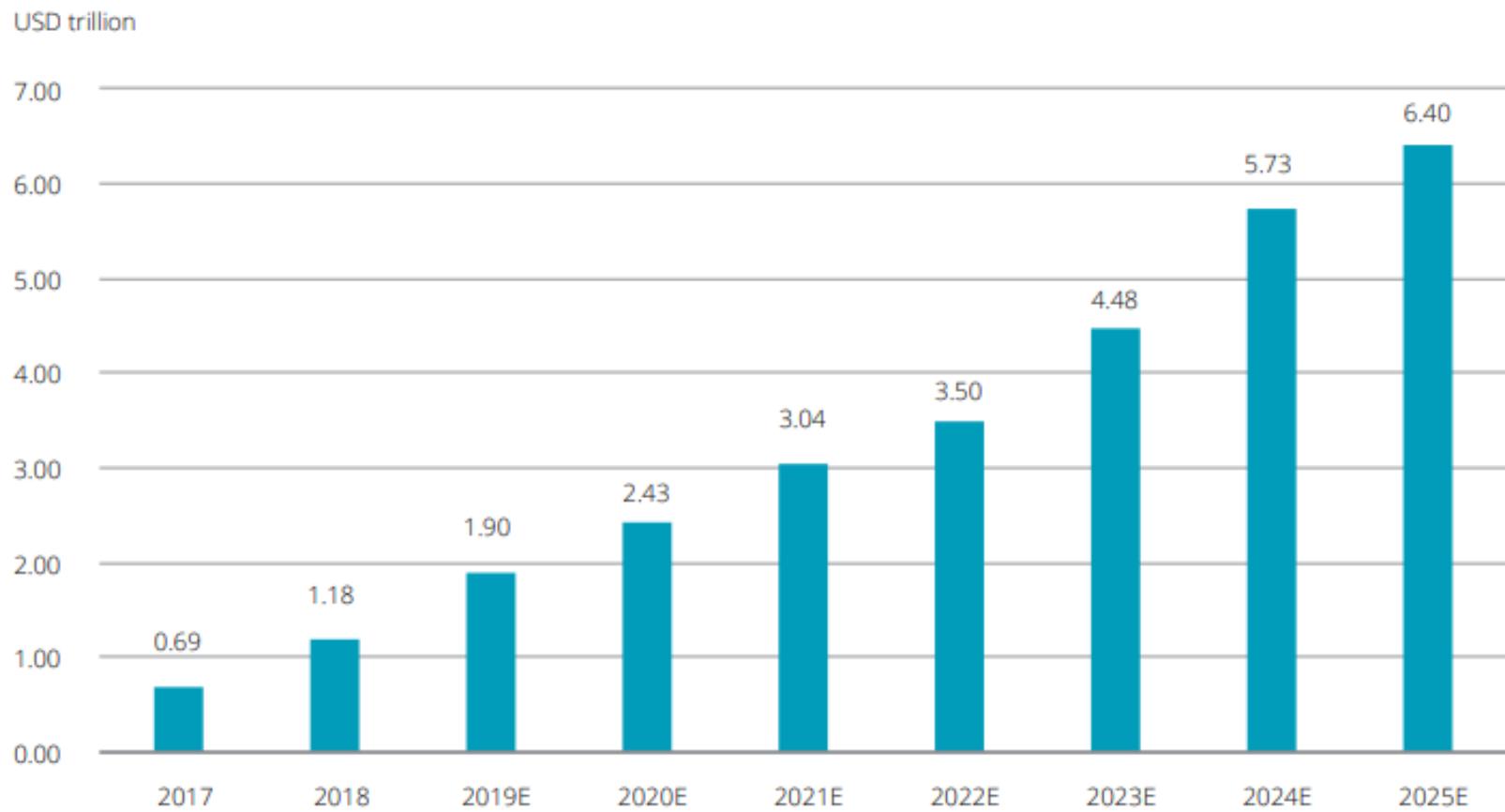
Un studiu al PricewaterhouseCoopers din 2019 estimează că PIB-ul global ar putea crește cu până la 14% până în 2030 ca rezultat al introducerii tehnologiei IA, în special pe baza cantității immense de date care se produce, de exemplu datele care se vor produce prin răspândirea pe scară largă a IoT



Estimarea câștigului pe baza introducerii IA în economie în diferite regiuni ale globului (Sursă: The macroeconomic impact of artificial intelligence, PricewaterhouseCoopers, 2018)



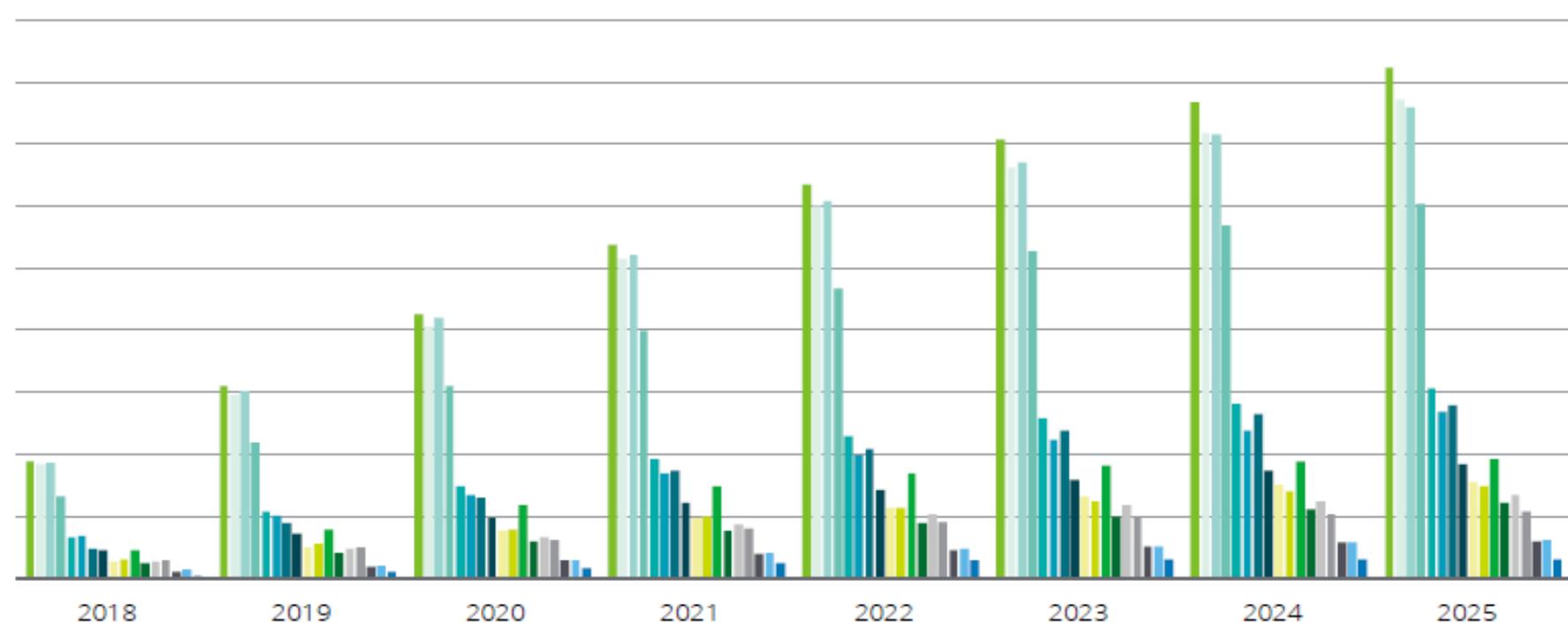
Conecțivitatea, disponibilitatea informației și inteligența artificială – a patra revoluție industrială (conform www.worldbank.org)



Source: Deloitte Research

Evoluția cifrei de afaceri a pieței mondiale de Inteligență Artificială
Conform Deloitte Research (2019)

USD10 billion



Source: Gartner

Cifra de afaceri pentru aplicațiile de IA în funcție de industrie
conform Deloitte Research (2019)

Leaderi din industrie creează un parteneriat pentru IA

- Amazon, DeepMind/Google, Facebook, IBM și Microsoft au creat o organizație non-profit pentru a promova înțelegerea tehnologiilor IA de către marele public și pentru a formula bune practici și oportunități în domeniul IA
- Partnership on Artificial Intelligence to Benefit People and Society

<https://www.partnershiponai.org/>

Inițiative ale Uniunii Europene

- **EU Initiative for Artificial Intelligence**
- **Trustworthy AI**
- **Confederation of EU AI laboratories**

<https://claire-ai.org/>

- **EU-Robotics**

<https://www.eu-robotics.net/>



Inteligentă Artificială

Universitatea Politehnica Bucuresti
Anul universitar 2020-2021

Adina Magda Florea



Curs 2

Strategii de căutare

- Reprezentarea soluției problemei
- Strategii de căutare de bază
- Strategii de căutare informate
- Strategii de căutare informate cu memorie limitată
- Determinarea funcției euristice

1 Reprezentarea soluției problemei

- Reprezentare prin spatiul starilor
- Reprezentare prin grafuri SI/SAU (AND-OR)
- Echivalenta reprezentarilor
- Caracteristicile mediului de rezolvare
- Pentru a prezenta si gasi o solutie:
 - Structura simbolica
 - Instrumente computationale
 - Metoda de planificare

1.1 Rezolvarea problemei reprezentată prin spațiul stărilor

- Stare
- Spatiu de stari
- Stare initială
- Stare/stari finală/finale
- (S_i, O, S_f)
- Solutia problemei
- Caracteristicile mediului

8-puzzle

S_i

| | | |
|---|---|---|
| 2 | | 3 |
| 1 | 8 | 4 |
| 7 | 6 | 5 |

S_f

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 8 | | 4 |
| 7 | 6 | 5 |



(a) Stare initială

(b) Stare finală

SUS - Mutare patrat liber in sus
 STINGA - Mutare patrat liber la stanga
 JOS - Mutare patrat liber in jos
 DREAPTA - Mutare patrat liber la dreapta

(c) Operatori

S_i

| | | |
|---|---|---|
| 2 | | 3 |
| 1 | 8 | 4 |
| 7 | 6 | 5 |

STINGA

JOS

DREAPTA

S_1

| | | |
|---|---|---|
| | 2 | 3 |
| 1 | 8 | 4 |
| 7 | 6 | 5 |

S_2

| | | |
|---|---|---|
| 2 | 8 | 3 |
| 1 | | 4 |
| 7 | 6 | 5 |

S_3

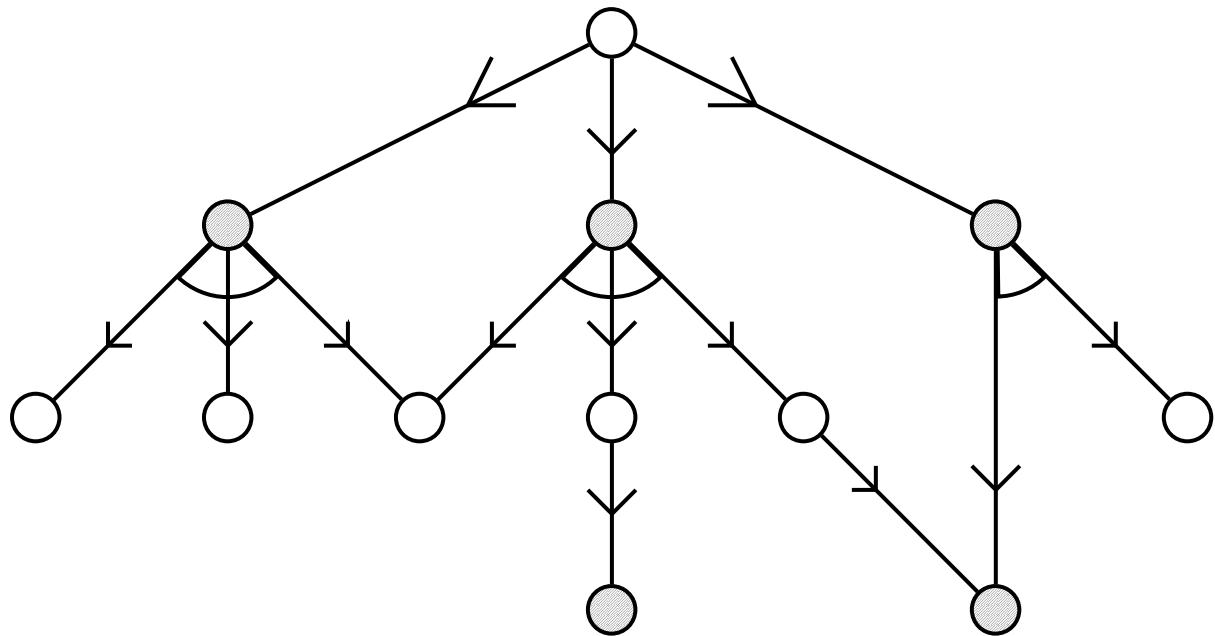
| | | |
|---|---|---|
| 2 | 3 | |
| 1 | 8 | 4 |
| 7 | 6 | 5 |

(d) Tranzitii posibile din starea S_i

1.2 Rezolvarea problemei reprezentată prin grafuri SI/SAU

- (P_i, O, P_e)
- Semnificatie graf SI/SAU (AND-OR)
- Nod rezolvat
- Nod nerezolvabil
- Solutia problemei

Graf SI/SAU

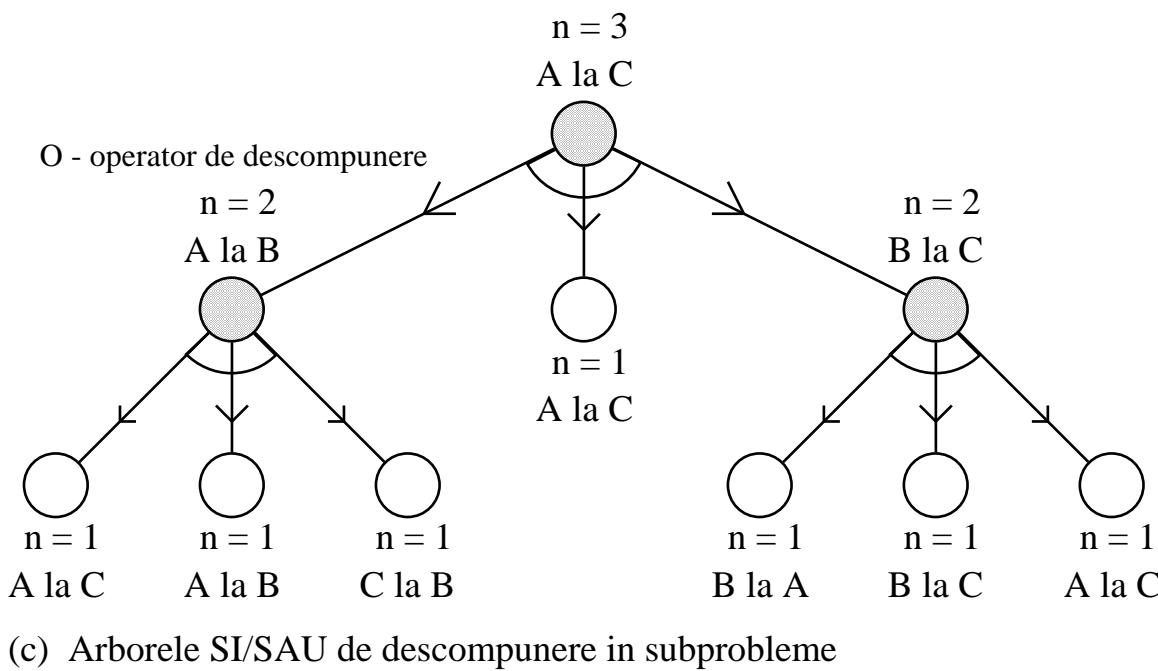
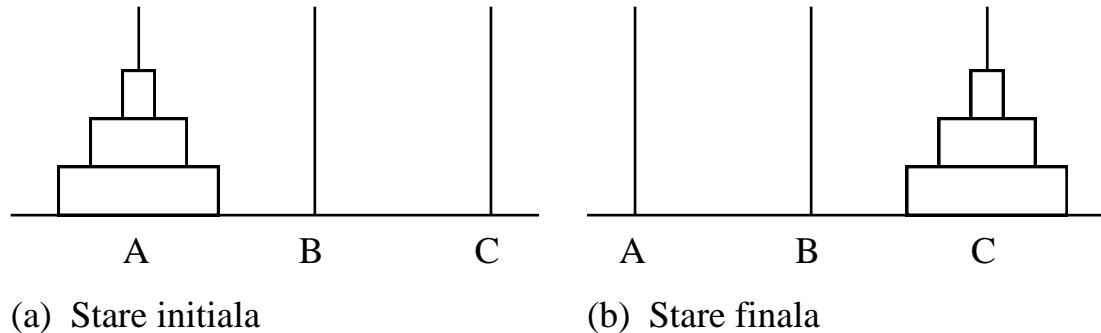


Nod SAU

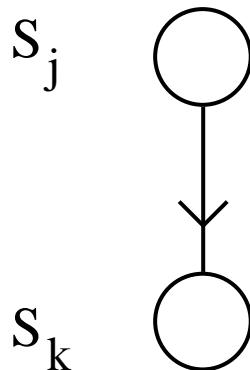
Noduri SI

Noduri SAU

Turnurile din Hanoi

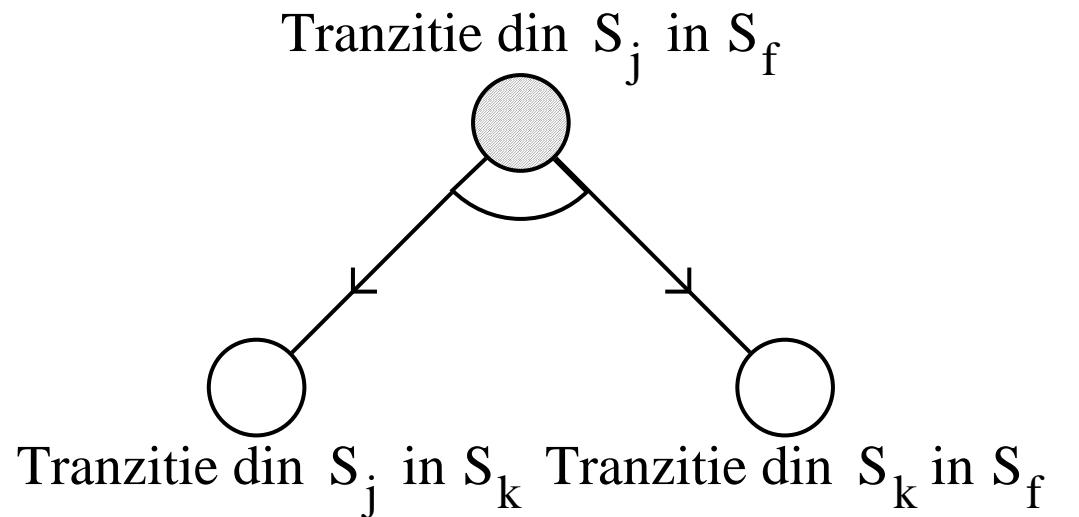


Echivalență reprezentărilor



S_j, S_k - stari intermediare S_f - stare finală

(a) Spatiul starilor



(b) Descompunerea problemei in subprobleme

1.3 Caracteristicile mediului

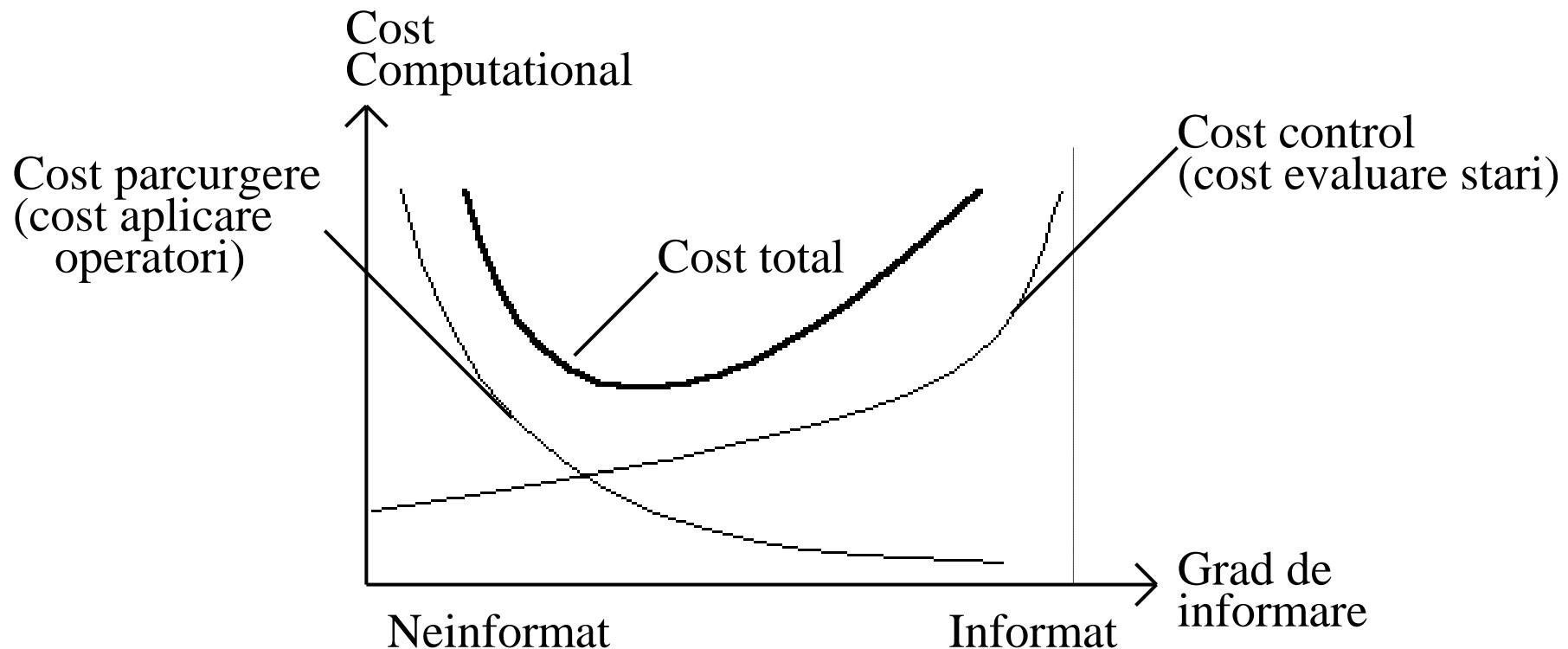
- Observabil / neobservabil
- Discret / continuu
- Finit / infinit
- Determinist / nedeterminist

2. Strategii de căutare de bază

Criterii de caracterizare

- Completitudine
- Optimalitate
- Complexitate: timp, spatiu
- Capacitatea de revenire
- Informare

Costuri ale căutării



2.1 Căutari neinformate

- Gasirea unei cai sau a tuturor cailor, cu cost sau fara cost
- **Cautarea pe nivel si cautarea in adancime** - parcurgerea se face in ordinea nodurilor succesoare starii curente in cautare:
 - Cautarea pe nivel – cele mai apropiate intai
 - Cautarea in adancime – cele mai departate intai
- **Algoritmul lui Dijkstra** rezolva problema celei mai scurte cai daca toate costurile arcelor sunt ≥ 0
- **Algoritmul Bellman-Ford** rezolva problema gasirii tuturor cailor de cost minim unde costurile arcelor pot fi si negative
- **Algoritmul Floyd-Warshall** rezolva problema gasirii tuturor cailor de cost minim

Căutari neinformate pentru grafuri implicit specificate

- Cautarea pe nivel (Breadth First Search - BFS)
- Cautare in adancime (Depth First Search - DFS)
- Cautare in adancime cu nivel iterativ (Iterative Deepening - ID)
- Cautare de tip backtracking
- Cautare bidirectională

Căutari neinformate în spațiul stărilor

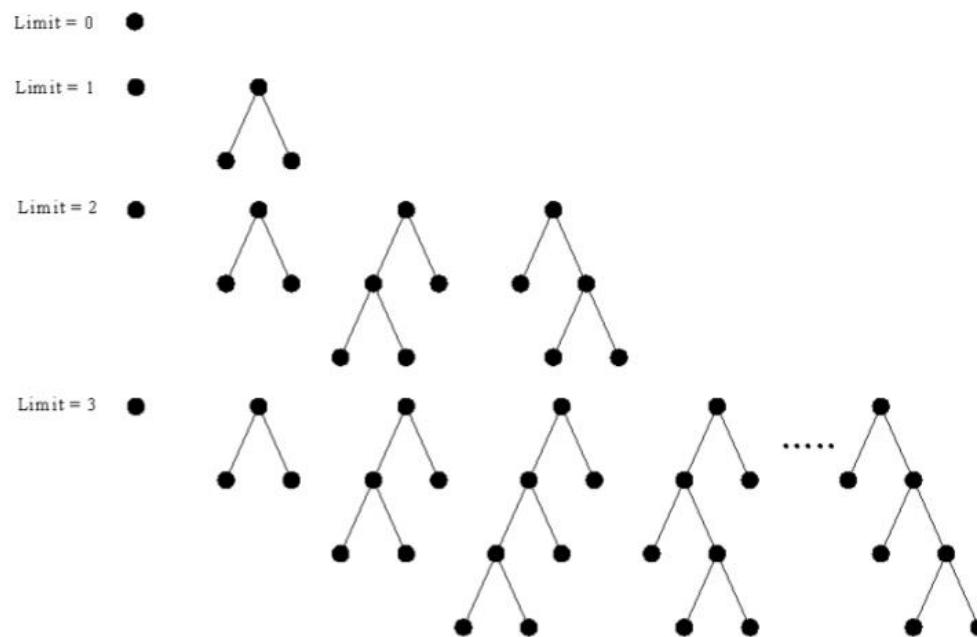
- Intr-o reprezentare a solutiei problemei prin spatiul starilor *adancimea unui nod* se defineste astfel:
 - $Ad(S_i) = 0$, unde S_i este nodul stare initiala,
 - $Ad(S) = Ad(S_p)+1$, unde S_p este nodul predecesor nodului S .
- Cele mai multe implementari bazate pe utilizarea a 2 liste: OPEN si CLOSED
- DFS – Open – stiva (LIFO)
- BFS – Open – coada (FIFO)
- In cele mai multe implementari CLOSED este implementata ca o tabela de dispersie (Hash)

Algoritm BFS / DFS(AdMax): Cautare pe nivel/in adancime in spatiul starilor

1. Initializeaza liste OPEN $\leftarrow \{S_i\}$, CLOSED $\leftarrow \{\}$
2. **daca** OPEN = {}
atunci intoarce INSUCES
3. Elimina primul nod S din OPEN si insereaza-l in CLOSED
4. **daca** $S \in OPEN \cup CLOSED$ (inainte de inserare S)
atunci repeta de la 2
- 4'. **daca** Ad(S) = AdMax **atunci repeta de al 2 /* pt DFS */**
5. Expandeaza nodul S
 - 5.1. Genereaza toti succesorii directi S_j ai nodului S
 - 5.2. **pentru** fiecare succesor S_j al lui S **executa**
 - 5.2.1. Stabileste legatura $S_j \rightarrow S$
 - 5.2.2. **daca** S_j este stare finala
atunci
 - i. Solutia este (S_j, \dots, S_i)
 - ii. **intoarce** SUCCES
 - 5.2.3. Insereaza S_j in OPEN, **la sfarsit / la inceput**
 6. **repeta de la 2**
sfarsit.

Căutări neinformate în spațiul stărilor

- Căutare în adincime cu nivel iterativ (ID)
pentru AdMax=1, m executa
DFS(AdMax)



Căutari neinformate în grafuri SI/SAU

- Se folosesc notiunile de nod rezolvat, nerezolvabil
- BFS, DFS
- Notiunea de adâncime a unui nod este diferita
- Într-o reprezentare a soluției problemei prin grafuri SI/SAU *adâncimea unui nod* se definește astfel:
 - $Ad(S_i) = 0$, unde S_i este nodul problema initială,
 - $Ad(S) = Ad(S_p) + 1$ dacă S_p este nod **SAU** predecesor al nodului S ,
 - $Ad(S) = Ad(S_p)$ dacă S_p este nod **SI** predecesor al nodului S .

Algoritm BFS-AND-OR: Cautare pe nivel in grafuri SI/SAU

1. Initializeaza listele $OPEN \leftarrow \{S_i\}$, $CLOSED \leftarrow \{\}$
2. Elimina primul nod S din $OPEN$ si insereaza-l in $CLOSED$
3. Expandeaza nodul S
 - 3.1. Genereaza toti succesorii directi S_j ai nodului S
 - 3.2. **pentru** fiecare succesor S_j al lui S **executa**
 - 3.2.1. Stabileste legatura $S_j \rightarrow S$
 - 3.2.2. **daca** S_j reprezinta o multime de cel putin 2 subprobleme
 - atunci** /* este nod SI */
 - i. Genereaza toti succesorii subprobleme S_j^k ai lui S_j
 - ii. Stabileste legaturile intre nodurile $S_j^k \rightarrow S_j$
 - iii. Insereaza nodurile S_j^k in $OPEN$, **la sfirsit**
 - 3.2.3. **altfel** insereaza S_j in $OPEN$, **la sfirsit**

4. **daca** nu s-a generat nici un succesor al lui S in pasul precedent
(3)

atunci

4.1. **daca** S este nod terminal etichetat cu o problema neelementara

atunci

4.1.1. Eticheteaza S nerezolvabil

4.1.2. Eticheteaza cu nerezolvabil toate nodurile predecesoare lui S care devin nerezolvabile datorita lui S

4.1.3. **daca** nodul S_i este nerezolvabil

atunci intoarce INSUCCES /* problema nu are solutie */

4.1.4. Elimina din OPEN toate nodurile care au predecesori nerezolvabili

4.2. altfel /* S este nod terminal etichetat cu o problema elementara */

4.2.1. Eticheteaza S rezolvat

4.2.2. Eticheteaza cu rezolvat toate nodurile predecesoare lui S care devin rezolvate datorita lui S

4.2.3. **daca** nodul S_i este rezolvat
atunci

i. Construieste arborele solutie urmarind legaturile

ii. **intoarce** SUCCES /* s-a gasit solutia */

4.2.4. Elimina din OPEN toate nodurile rezolvate si toate nodurile care au predecesori rezolvati

5. repeta de la 2
sfarsit.

2.2 Complexitatea strategiilor de căutare

- B - *factorul de ramificare* al unui spatiu de cautare 8-puzzle
- Numar de miscari:
- 2 m pt colt = 8
- 3 m centru lat = 12
- 4m centru $\Rightarrow 24$ miscari
- **$B = nr. misc. / nr. poz. p. liber = 2.67$**
- Numar de miscari:
- 1 m pt colt = 4
- 2 m centru lat = 8
- 3m centru $\Rightarrow 15$ miscari $\Rightarrow B = 1.67$

Complexitatea strategiilor de căutare

- B - *factorul de ramificare*
- d - adancimea celui mai apropiat nod solutie (de cost minim daca exista costuri)
- m – lungimea maxima a oricarei cai din spatiul de cautare

Rad – B noduri, B^2 pe niv 2, etc.

- Numarul de stari posibil de generat pe un nivel de cautare d este B^d
- T - numarul total de stari generate intr-un proces de cautare, d – adancime nod solutie

$$T = B + B^2 + \dots + B^d = O(B^d)$$

Complexitatea strategiilor de căutare

- **Cautare pe nivel**

Numar de noduri generate

$$B + B^2 + \dots + B^d = O(B^d)$$

- **Cautare in adancime**

Numar de noduri generate - B^*d – daca nodurile expandate se sterg din CLOSED

Sau B^*m cu AdMax=m

Complexitatea strategiilor de căutare

- **Cautare backtracking**

Numar de noduri generate m – daca se elimina
CLOSED

- **Cautare in adancime cu nivel iterativ**

Numar de noduri generate

$$d * B + (d-1) * B^2 + \dots + (1) * B^d = O(B^d)$$

Complexitatea strategiilor de căutare

| Criteriu | Nivel | Adancime | Adanc. limita | Nivel iterativ | Bidirectionala |
|---------------|-------|----------|--------------------|----------------|----------------|
| Timp | B^d | B^d | B^m | B^d | $B^{d/2}$ |
| Spatiu | B^d | B^*d | B^*m | B^d | $B^{d/2}$ |
| Optimalitate? | Da | Nu | Nu | Da | Da |
| Completa? | Da | Nu | Da daca $m \geq d$ | Da | Da |

B – factor de ramificare, **d** – adancimea solutiei,
m – adancimea maxima de cautare (AdMax)

3. Strategii de căutare informate

Cunoștințele euristice pot fi folosite pentru a crește eficiența căutării în trei moduri:

- Selectarea nodului urmator de expandat în cursul căutării
- În cursul expandării unui nod al spațiului de căutare se poate decide pe baza informațiilor euristice care dintre succesorii lui vor fi generati și care nu
- Eliminarea din spațiul de căutare a anumitor noduri generate

3.1 Căutare informata de tip "best-first"

- Evaluarea cantitatii de informatie
- Calitatea unui nod este estimata de *functia de evaluare euristica*, notata $w(n)$ pentru nodul n
- Presupuneri pentru functia $w(n)$
- Exemple
 - Strategia de cautare a alpinistului
 - Strategia de cautare “best-first”

Algoritm BestFS: Cautare "best-first" in spatiul starilor

Intrari: Starea initiala S_i si functia $w(S)$ asociata starilor

Iesiri: SUCCES si solutia sau INSUCCES

1. Initializeaza listele $OPEN \leftarrow \{S_i\}$, $CLOSED \leftarrow \{\}$
2. Calculeaza $w(S_i)$ si asociaza aceasta valoare nodului S_i
3. **daca** $OPEN = \{\}$
atunci intoarce INSUCCES
4. Elimina nodul S cu $w(S)$ minim din $OPEN$ si insereaza-l in $CLOSED$
5. **daca** S este stare finala
atunci
 - i. Solutia este (S, \dots, S_i)
 - ii. **intoarce** SUCCES
6. Expendeaza nodul S
 - 6.1. Genereaza toti succesorii directi S_j ai nodului S

6.2. pentru fiecare succesor S_j al lui S **executa**

- 6.2.1 Calculeaza $w(S_j)$ si asociaza-l lui S_j**
- 6.2.2. Stabileste legatura $S_j \rightarrow S$**
- 6.2.3. daca $S_j \notin OPEN \cup CLOSED$**
atunci introduce S_j in OPEN cu $w(S_j)$ asociat
- 6.2.4. altfel**

- i. Fie S'_j copia lui S_j din OPEN sau CLOSED
- ii. **daca $w(S_j) < w(S'_j)$**
atunci
 - Elimina S'_j din OPEN sau CLOSED
(de unde apare copia)
 - Insereaza S_j cu $w(S_j)$ asociat in OPEN
- iii. **altfel** ignora nodul S_j

7. repeta de la 3

sfarsit.

Cazuri particulare

- Strategia de cautare "best-first" este o generalizare a strategiilor de cautare neinformate
 - strategia de cautare pe nivel $w(S) = Ad(S)$
 - strategia de cautare in adincime $w(S) = -Ad(S)$
- Strategia de cautare de cost uniform / Dijkstra

$$w(S_j) = \sum_{k=i}^{j-1} \text{cost_arc}(S_k, S_{k+1})$$

- Minimizarea efortului de cautare – cautare euristica
 - $w(S) = \text{functie euristica}$

3.2 Căutarea soluției optime în spațiul starilor. Algoritmul A*

w(S) devine **f(S)** cu 2 comp:

- **g(S)**, o functie care estimeaza costul real $g^*(S)$ al caii de cautare intre starea initiala S_i si starea S,
- **h(S)**, o functie care estimeaza costul real $h^*(S)$ al caii de cautare intre starea curenta S si starea finala S_f .
- $f(S) = g(S) + h(S)$
- $f^*(S) = g^*(S) + h^*(S)$

Calculul lui f(S)

- Calculul lui g(S)

$$g(S) = \sum_{k=i}^n \text{cost_arc}(S_k, S_{k+1})$$

- Calculul lui h(S)
- Trebuie sa fie admisibila
- O functie euristica h se numeste *admisibila* daca pentru orice stare S, $h(S) \leq h^*(S)$ & $h(S_f)=0$
- Definitia stabileste ***conditia de admisibilitate*** a functiei h si este folosita pentru a defini ***proprietatea de admisibilitate*** a unui algoritm A*.

A* - proprietatea de admisibilitate

Fie un algoritm A* care utilizeaza cele doua componente **g** si **h** ale functiei de evaluare **f**. Daca

- (1) functia **h** satisface conditia de admisibilitate
- (2) $\text{cost_arc}(S, S') \geq c$

pentru orice doua stari S, S' , unde $c > 0$ este o constanta si costul **c** este finit

atunci **algoritmul A* este admisibil**, adica este garantat sa gaseasca calea de cost minim spre solutie.

- Completitudine – garantat sa gaseasca solutie daca solutia exista si costurile sunt pozitive

$h(S)$ aproape de $h^*(S)$?

- Fie doi algoritmi A*, A1 si A2, cu functiile de evaluare h_1 si h_2 admisibile, $g_1=g_2$

$$f_1(S) = g_1(S) + h_1(S) \quad f_2(S) = g_2(S) + h_2(S)$$

- Se spune ca algoritmul **A2 este mai informat decat** algoritmul **A1** daca pentru orice stare S cu $S \neq S_f$

$$h_2(S) > h_1(S)$$

h_2 domina h_1

$h(S)$ monotonă?

- **Monotonia functiei $h(S)$**

Daca

$$h(S) \leq h(S') + \text{cost_arc}(\text{op}, S, S') \quad \& \quad h(S_f) = 0$$

pentru orice doua stari S si S' succesor al lui S , cu S' diferit de S_f , din spatiul de cautare

atunci se spune ca $h(s)$ este **monotona (sau consistenta)**

- Daca **h este monotonă** atunci avem garantia ca un nod introdus in CLOSED nu va mai fi niciodata eliminat de acolo si reintrodus in OPEN iar implementarea se poate simplifica corespunzator

Implementare A*

Strategia de cautare "best-first" se modifica:

...

2. Calculeaza $w(S_i) = g(S_i) + h(S_i)$ si asociaza aceasta valoare nodului S_i
3. **daca** $OPEN = \{\}$

atunci intoarce INSUCCES - *nemodificat*

4. Elimina nodul S cu $w(S)$ minim din $OPEN$ si insereaza-l in $CLOSED$ - *nemodificat*

.....

6.2.4. **altfel**

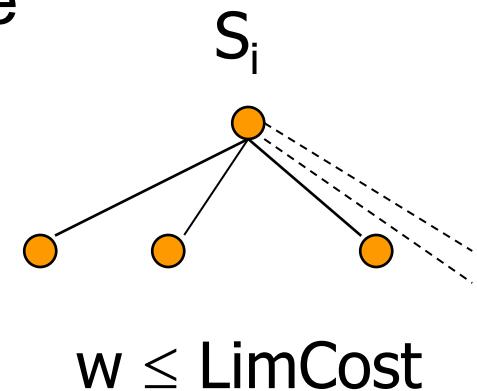
- i. Fie S'_j copia lui S_j din $OPEN$ sau $CLOSED$
 - ii. **daca** $g(S_j) < g(S'_j)$
- atunci ...

4. Strategii de căutare informată cu memorie limitată

- A* se termina intotdeauna gasind o solutie optima si poate fi aplicat pe probleme generale
- Cu toate acestea, cantitatea de memorie necesara creste repede pe masura avansului algoritmului.
- Presupunem 100 bytes pt a memora o stare si atributele ei; rezulta aproximativ 10 mbytes/sec => o memorie de 1G se utilizeaza in mai putin de 2 minute
- Algoritmii clasici pentru cautare cu memorie limitata sunt:
 - Depth first iterative deepening (DFID)
 - Iterative deepening A* (IDA*)

DFID cu cost

- Cautarea realizeaza BFS cu o serie de DFS care opereaza pe o frontiera de cautare care creste succesiv
- Cautarea in adancime este modificata a.i. sa utilizeze o **limita de cost** in loc de o limita a adancimii
- Fiecare iteratie expandeaza nodurile din interiorul unui **contur de cost** pentru a vedea care sunt nodurile de pe urmatorul contur
- Daca cost arce 1 atunci DFID fara cost



DFID cu cost

- Algoritmul utilizeaza **doua limite U si U'** pentru urmatoarea iteratie.
- Apeleaza repetitiv functia DFID care cauta o cale de cost minim **p**.
- DFID actualizeaza **variabila globala U'** la **valoarea minima a costului cailor** generate pana intr-un moment al cautarii
- Daca spatiul de cautare nu contine solutia si este infinit, algoritmul nu se termina

Algoritm DFID: Depth first iterative deepening cu cost

Foloseste

- functia iterativa **BuclaDFID**
- functia recursiva **DFID**
- Functia **Expand** pentru generarea succesorilor unui nod
- Functia **Goal** care testeaza daca stare finala

BuclaDFID

Intrari: Starea initiala s si functia de cost w(s) asociata starilor

Iesiri: Calea de la s la starea finala sau {}

$U' \leftarrow 0$, $\text{bestPath} \leftarrow \{\}$ /* initializare limita globala si cale */

cat timp ($\text{bestPath} = \{\}$ si $U' \neq \text{inf}$) **repeta** /* nu s-a gasit solutie, exploreaza */

$U \leftarrow U'$, $U' \leftarrow \text{inf}$ /* reset limita, init limita globala noua */

$\text{bestPath} \leftarrow \text{DFID}(s, 0, U)$

intoarce bestPath

sfarsit

DFID(s, g, U)

Intrari: starea s, costul caii g, limita superioara U

lesiri: calea de la s la starea finală sau {}

Efect lateral: Actualizarea lui U'

daca (Goal(s)) atunci intoarce Cale(s)

$\text{Suc}(s) \leftarrow \text{Expand}(s)$

pentru fiecare v in Suc(s) repeta

atunci $p \leftarrow \text{DFID}(v, g+w(s,v), U)$

daca $p \neq \emptyset$ **atunci** **intoarce** (p) /* s-a gasit solutie */

altfel /* cost mai mare decat limita veche U */

daca g + w(s,v) < U' /* cost mai mic decat limita globala */

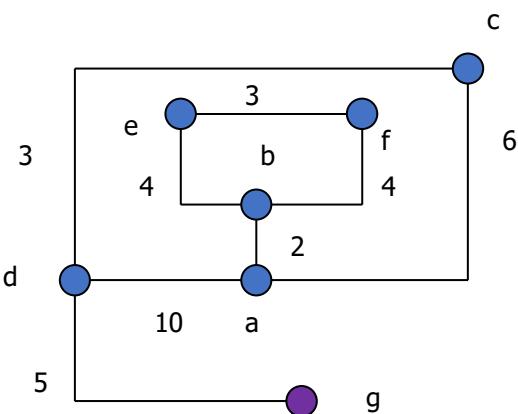
atunci $U' \leftarrow g + w(s, v)$ /* seteaza limita globala noua */

intoarce {

sfarsit

| Pas | Iteratie | Selectie | Apeluri | U | U' | Obs | Cautare DFID |
|-----|----------|----------|---------------------|---|-----|--------------------------|--------------|
| 1 | 1 | {} | {(a,0)} | 0 | inf | | |
| 2 | 1 | a | {} | 0 | 2 | g(b), g(c) si $g(d) > U$ | |
| 3 | 2 | {} | {(a,0)} | 2 | inf | Incepe o noua iteratie | |
| 4 | 2 | a | {(b,2)} | 2 | 6 | g(c) si $g(d) > U$ | |
| 5 | 2 | b | {} | 2 | 6 | g(e) si $g(f) > U$ | |
| 6 | 3 | {} | {(a,0)} | 6 | inf | Incepe o noua iteratie | |
| 7 | 3 | a | {(b,2),(c,6)} | 6 | 10 | $g(d) > U$ | |
| 8 | 3 | b | {(e,6),(f,6),(c,6)} | 6 | 10 | | |
| 9 | 3 | e | {(f,6),(c,6)} | 6 | 10 | $g(f) > U$ | |
| 10 | 3 | f | {(c,6)} | 6 | 10 | $g(e) > U$ | |
| 11 | 3 | c | {} | 6 | 9 | $g(d)$ | |
| 12 | 4 | {} | {(a,0)} | 9 | inf | Incepe o noua iteratie | |

.....



pentru fiecare v in $Suc(s)$ repeta

daca $g + w(s,v) \leq U$

atunci $p \leftarrow DFID(v, g+w(s,v), U)$

daca $p \neq \{ \}$ atunci intoarce (p)

altfel

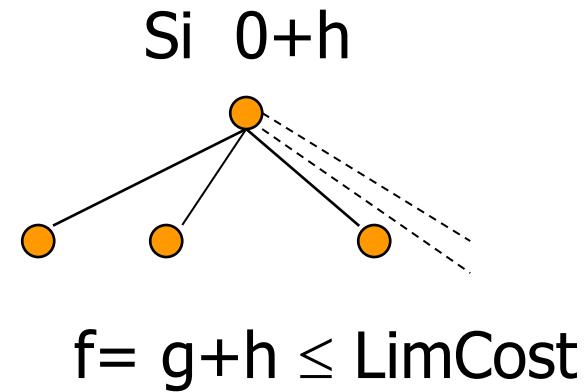
daca $g + w(s,v) < U'$ atunci $U' \leftarrow g + w(s,v)$

cat timp ($bestPath = \{ \}$ si $U' \neq inf$) repeta

$U \leftarrow U'$, $U' \leftarrow inf$
 $bestPath \leftarrow DFID(s, 0, U)$

IDA*

- Iterative deepening A*
- Bazat pe DFID cu cost
- Garantat sa gaseasca solutia de cost minim
- $f(s) = g(s) + h(s)$



Algoritm IDA*: Iterative deepening A*

Foloseste

- functia iterativa **BuclaIDA***
- functia recursiva **IDAg***
- Functia **Expand** pentru generarea succesorilor unui nod
- Functia **Goal** care testeaza daca stare finala

BuclaIDA*

Intrari: Starea initiala s, functia de cost w(s) si h euristica asociata starilor

Iesiri: Calea de la s la starea finala sau {}

$U' \leftarrow h(s)$, bestPath $\leftarrow \{ \}$

cat timp (bestPath = {} si $U' \neq \text{inf}$) **repeta**

$U \leftarrow U'$, $U' \leftarrow \text{inf}$

 bestPath $\leftarrow \text{IDA}^*(s, 0, U)$

intoarce bestPath

sfarsit

IDA*(s, g, U)

Intrari: starea s, costul caii g, limita superioara U

Iesiri: calea de la s la starea finala sau {}

Efect lateral: Actualizarea lui U'

daca (Goal(s)) **atunci intoarce** Cale(s)

Suc(s) \leftarrow Expand(s)

pentru fiecare v in Suc(s) **repeta**

daca $g + w(s, v) + h(v) \leq U$ /* starea este in limita U */

atunci

$p \leftarrow \text{IDA}^*(v, g + w(s, v), U)$

daca $p \neq \{\}$ **atunci intoarce** p /* s-a gasit solutie */

altfel /* cost mai mare decat limita veche U */

daca $g + w(s, v) + h(v) < U'$ /* cost mai mic decat noua limita */

atunci $U' \leftarrow g + w(s, v) + h(v)$ /* actualizez noua limita */

intoarce {}

sfarsit

5. Determinarea funcției de evaluare h

- Specifica problemei, determinate manual (hand crafted)

SAU

- Generare automata de euristici
- **Transformare abstractă** prin relaxarea unor restrictii ale problemei
- Pattern databases – precalculeaza si memoreaza distanta pana la solutie intr-un spatiu abstract generat de relaxarea restrictiilor impuse miscarilor/actiunilor
- Ne intereseaza o functie euristica $h(s)$ cat mai aproape de $h^*(s)$
- Am dori si un effort cat mai mic

Exemple de funcții euristice

- **Problema comis-voiajorului**

$$h_1(S) = \text{cost_arc}(S_i, S)$$

- $h_2(S) = \text{costul arborelui de acoperire de cost minim al oraselor neparcurse pana in starea } S$

Exemple de funcții euristice

- 8-puzzle

$$h_1(S) = \sum_{i=1}^8 t_i(S)$$

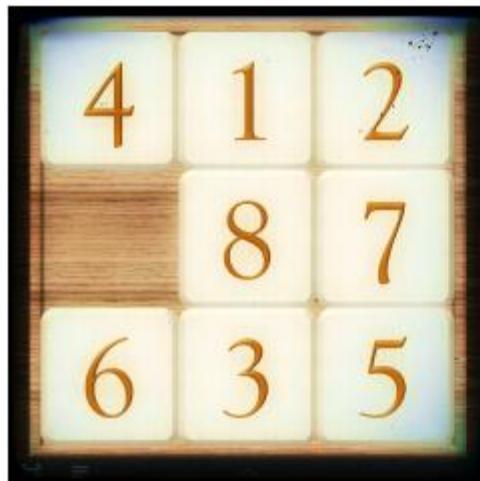
$$h_2(S) = \sum_{i=1}^8 \text{Distanta}(t_i)$$

Distanta Manhattan

$$dx = |\text{nod.x}-\text{scop.x}|$$

$$dy = |\text{nod.y}-\text{scop.y}|$$

$$h(\text{nod}) = (dx+dy)$$



Relaxarea condiției de optimalitate a algoritmului A*

- O functie euristică h se numeste **ε -admisibila** daca
$$h(S) \leq h^*(S) + \varepsilon \quad \text{cu } \varepsilon > 0$$
- Algoritmul A* care utilizeaza o functie de evaluare f cu o componenta h ε -admisibila gaseste intotdeauna o solutie al carei cost depaseste costul solutiei optime cu cel mult ε .
- Un astfel de algoritm se numeste *algoritm A* ε -admisibil* iar solutia gasita se numeste *solutie ε -optimala*.

Relaxarea condiției de optimalitate a algoritmului A*

- 8-puzzle

$$f_3(S) = g(S) + h_3(S) \quad h_3(S) = h_2(S) + 3 \cdot T(S)$$

$$T(S) = \sum_{i=1}^8 \text{Scor}[t_i(S)]$$

$$\text{Scor}[t_i(S)] = \begin{cases} 2 & \text{daca patratul } t_i \text{ in starea } S \text{ nu este urmat de} \\ & \text{succesorul corect din starea finala} \\ 0 & \text{pentru orice pozitie a lui } t_i \text{ diferita de centru} \\ 1 & \text{pentru } t_i \text{ aflat la centrul mozaicului} \end{cases}$$

Transformari abstracte pt functii euristice

- O transformare abstracta $F:S \rightarrow S'$, spatiu abstract, mapeaza stari **s** din S (problema reala) in stari $F(s)$ din S' si mutari/actiuni **a** din problema reala in actiuni in spatial abstract
- Daca distanta intre oricare 2 stari $F(u)$ si $F(v)$ in spatial abstract este mai mica sau egala decat distanta intre u si v , atunci aceasta distanta poate fi utilizata ca o euristică admisibila
- Calculate pe parcursul cautarii sau stocate in pattern databases

Transformari abstracte pt functii euristice

- Variante “relaxate” ale problemei
- $h1$ si $h2$ din 8 puzzle reprezinta de fapt distante dintr-o versiune simplificata a problemei
 - (1) O piesa poate fi mutata de la A la B daca B este liber
 - (2) O piesa poate fi mutata de la A la B daca A si B sunt adiacente

Variante “relaxate” ale problemei

O piesă poate fi mutată de la A la B dacă A și B sunt adiacente orizontal sau vertical și B este liber

- (a) O piesă poate fi mutată de la A la B dacă A și B sunt adiacente .
- (b) O piesă poate fi mutată de la A la B dacă B este liber .
- (c) O piesă poate fi mutată de la A la B .

Cum putem găsi o funcție euristică?

- A* în jocuri pt parcurgerea unui teritoriu

Distanta Manhattan

$$dx = |\text{nod}.x - \text{scop}.x|$$

$$dy = |\text{nod}.y - \text{scop}.y|$$

$$h(\text{nod}) = (dx + dy)$$

Distanta Euclidiană

$$dx = |\text{nod}.x - \text{scop}.x|$$

$$dy = |\text{nod}.y - \text{scop}.y|$$

$$h(\text{nod}) = \sqrt{dx^2 + dy^2}$$

Cum putem găsi o funcție euristică?

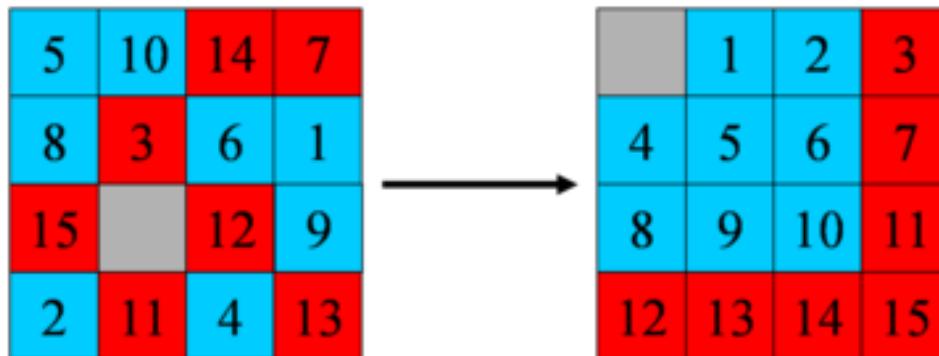
- **Pattern database pt euristici**
- Memoreaza o colectie de solutii ale unor subprobleme care trebuie rezolvate pentru a rezolva problema
- Fiecare solutie de subproblemă are o functie euristică precalculată (costul căutării) și memorată
- **Pattern** – o specificare parțială a unei stări
- **Target pattern** – o specificare parțială a stării scop

Pattern database pt euristici

- **Pattern database** – multimea tuturor pattern-urilor care pot fi obtinute prin relaxari sau permutari ale target pattern
- Pentru fiecare pattern din baza de date calculam distanta (nr minim de mutari) fata de target pattern folosind analiza inversa.
- Distanta este costul pattern-ului

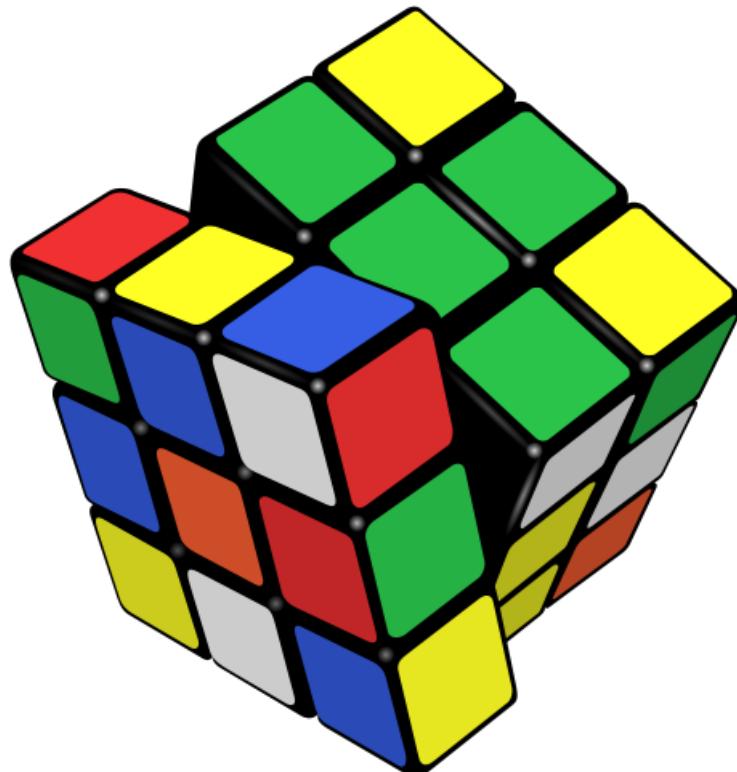
Pattern database pt euristici

- 15 puzzle
- Baza de date va indica numarul minim de mutari necesare pt a duce la locul bun 3, 7, 11, 12, 13, 14 si 15; apoi se rezolva 8-puzzle (albastru)
- 31 mutari pt a rezolva piesele rosii (22 mutari pentru a rezolva piesele albastre)



Cubul lui Rubik

- 9 patrate cu 6 culori diferite
- Cea mai buna solutie IDA*



Cubul lui Rubik

Euristici

- ***Distanta Manhattan 3D*** = Calculeaza distanta liniara intre 2 puncte in R3 prin insumarea distantelelor punctului in fiecare dimensiune
- Distanta Manhattan 3D intre punctele p1 si p2
$$MD3d(p1, p2) = |x_1 - x_2| + |y_1 - y_2| + |z_1 - z_2|$$
- Poate fi calculata in timp liniar
- Trebuie impartita la 8 pentru a fi admisibila deoarece fiecare miscare muta 4 colturi si 4 muchii

Cubul lui Rubik

- *Dureaza mult*
- Adancime 18 – aprox 100 ani
- **Pattern database**
- Se memoreaza intr-o tabela numarul de miscari necesare pt a rezolva colturile cubului sau subprobleme

Cea mai buna?

- Avem mai multe euristici bune
- Pe care o alegem?
- $h(n) = \max (h_1(n), \dots, h_k(n))$

Inteligentă Artificială

Universitatea Politehnica Bucuresti
Anul universitar 2020-2021

Adina Magda Florea



Curs 3

Strategii de cautare

- Cautari locale
- Cautari on-line
- Problema satisfacerii restrictiilor (CSP)

1. Cautari locale

- **Cautari locale** – opereaza asupra starii curente generand succesorii
- Calea catre solutie nu are importanta
- Gasire solutie – CSP - nu conteaza calitatea solutiei
- Gasire solutie optima – functie de evaluare sau de cost
- Probleme in gasirea solutiei optime pe baza de cautari locale in functie de forma spatiului de cautare
- Maxim global, maxim local, platou, umar

Cautari locale

Caracteristici

- Folosesc putina memorie
- Gasesc solutii destul de bune chiar si in spatii infinite
- Folosite si pt probleme de optimizare
- Hill climbing
- Simulated annealing

Algoritm: **Hill climbing**

/* intoarce o stare cu functia de evaluare un maxim local */

1. $S \leftarrow$ stare initiala
2. Genereaza $S_j = \text{Succ}(S)$, toti succesorii starii S
3. $S' \leftarrow S_j$ cu $\text{Eval}(S_j)$ maxim dintre toti succesorii S_j
4. **daca** $\text{Eval}(S') \leq \text{Eval}(S)$ **atunci intoarce** S
5. $S \leftarrow S'$
6. **repeta** de la 2
sfarsit

- Cautarea se orienteaza permanent in directia cresterii valorii;
- Se termina cand ajunge la un maxim (nici un succesor nu are valoarea mai mare)

Hill Climbing

- Problema 8 regine
- S – tabla completa
- Succesori – toate starile posibil de generat prin mutarea a 1 regina intr-un alt patrat in coloana
- o stare are $8 \times 7 = 56$ succesorii
- Fct de evaluare = nr de perechi de regine care se ataca
- Minim global = 0
- 86% instante nu gaseste solutia (3 pasi)
- 14% gaseste soluta (4 pasi)
- Spatiul starilor $8^8 \approx 17$ milioane de stari

Hill Climbing - imbunatatiri

- Miscari laterale – se spera sa fie “umar” si nu “platou” (continui si pentru $\text{Eval}(S') = \text{Eval}(S)$)
- Daca “platou” – risc de bucla infinita – necesita limita in cautare
- Cu aprox. 100 miscari laterale – problema 8 regine – 94% instante pt care se gaseste solutia

Hill Climbing - imbunatatiri

Hill climbing stochastic

- Dintre starile succesoare cu $\text{Eval}(S_j) \geq \text{Eval}(S)$, se alege aleator un S_j

First choice hill climbing

- Genereaza aleator succesiuni pana gaseste $\text{Eval}(S_j) \geq \text{Eval}(S)$, continua cautarea cu S_j

Random restart Hill Climbing

- Repeta diferite HC cu stari initiale generate aleator
- Daca fiecare HC are o probabilitate de succes de $p \rightarrow 1/p$ repetitii

Simulated annealing

- Simuleaza un proces fizic (calirea)
- T – temperatura
- Scaderea gradientului – solutii de cost minim
- Alege o miscare la intamplare
- Daca starea este mai buna, cauta in continuare de la aceasta
- Altfel alege starea mai "proasta" cu o probabilitate
- Scade probabilitatea de selectie a starilor mai "proaste" pe masura ce temperatura scade

Algoritm: Cautare Simulated Annealing

1. $T \leftarrow$ temperatura initiala
2. $S \leftarrow$ stare initiala
3. $v \leftarrow \text{Eval}(S)$
2. **cat timp** $T >$ temp finala **executa**
 - 2.1 **pentru** $i=1,n$ **executa**
 - $S' \leftarrow \text{Succ}(S)$
 - $v' \leftarrow \text{Eval}(S')$
 - daca** $v' < v$ **atunci** $S \leftarrow S'$
 - altfel** $S \leftarrow S'$ cu prob. $\exp(-(v'-v)/T)$
(in rest S nemodificat)
 - 2.2 $T \leftarrow 0.95 * T$
- sfarsit**

Cautari locale in spatii continue

- Factor de ramificare infinit
- First choice HC, Simulated annealing
- **Problema:** dorim sa plasam 3 supermarket-uri pe o harta a.i. suma patratelor distantele de la fiecare comuna de pe harta la cel mai apropiat supermarket sa fie minima.
- Spatiul starilor este definit prin coordonatele supermarket-urilor

$$(x_1, y_1) (x_2, y_2) (x_3, y_3)$$

- (spatiu n -dimensional cu n variabile)
- O miscare in acest spatiu – miscarea unui sm pe harta
- C_i - multimea de orase care sunt cel mai aproape de sm i in starea curenta

$$f(x_1, y_1, x_2, y_2, x_3, y_3) = \sum_{i=1,3} \sum_{c \in C_i} (x_i - x_c)^2 + (y_i - y_c)^2$$

Cautari locale in spatii continue

Cum putem gasi solutia?

VARIANTE

- Discretizare spatiu – discretizarea vecinatatii fiecarei stari (de ex mutam cate un sm o data fie in directa x fie in directia y cu cate o cantitate fixa $M \Rightarrow 12$ stari succesoare)
- Utilizarea gradientului pt a gasi maximul
- Determinam gradientul lui f , global sau local

Si aplicam HC prin actualizarea starii curente cu $x + constanta * gradient$

2. Cautare on-line

- Cautare “offline”
- **Cautare “on-line” – cautare + actiune**
- Exemplu: robot care investigheaza mediul
- **Cautare online:** pentru fiecare actiune, agentul primeste perceptia care ii spune in ce stare a ajuns.
Construieste Rezultat[s,a]
- **Cautari locale online:**
 - HC, nu pot random restart,
 - Adauga memorie la HC – memoreaza cea mai buna estimare curenta

Cautare on-line

- Programare dinamica asincrona
- Learning Real-Time A*
- Cautare cu tinta mobila

2.1 Programare dinamica asincrona (ADP)

Principiul optimalitatii – o cale este optima \leftrightarrow orice segment (subcale) a acesteia este optima

- $S_i \rightarrow S_f$ si S pe aceasta cale
- $S_i \rightarrow S, S \rightarrow S_f$

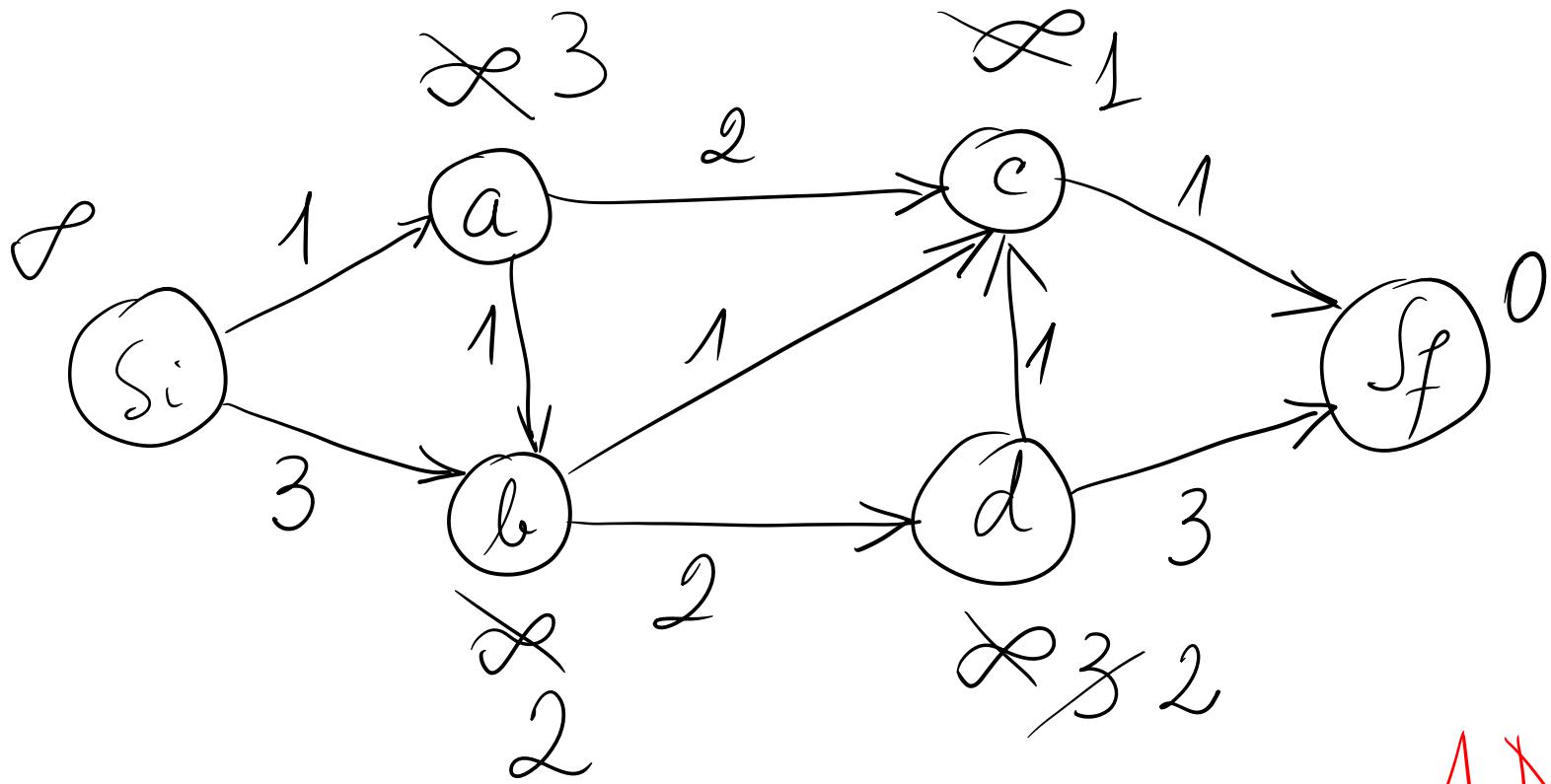
Daca se cunoaste h^* pt fiecare nod, calea de cost minim poate fi obtinuta repetand procedura:

- Pentru fiecare nod succesor j a nodului curent i calculeaza $f^*(j)=c(i,j) + h^*(j)$
- Mergi la starea j pt care $f^*(j)$ este minim

Programare dinamica asincrona

Presupunem urmatoarea situatie

- Pentru fiecare nod i exista un proces care corespunde lui i
- Fiecare proces inregistreaza $h(i)$ – estimarea lui $h^*(i)$
- Valoarea initiala a lui $h(i)$ este arbitrara cu exceptia nodurilor stare finala
- Fiecare proces i actualizeaza $h(i)$:
 - pentru fiecare vecin j - calculeaza $f(j)=c(i,j) + h(j)$
 - $h(i) \leftarrow \min_j f(j)$



ADP

$$f(j) = c(i,j) + h(j)$$

Programare dinamica asincrona

Ordinea de actualizare este arbitrara

Daca costul arcelor este pozitiv, converge la valorile reale

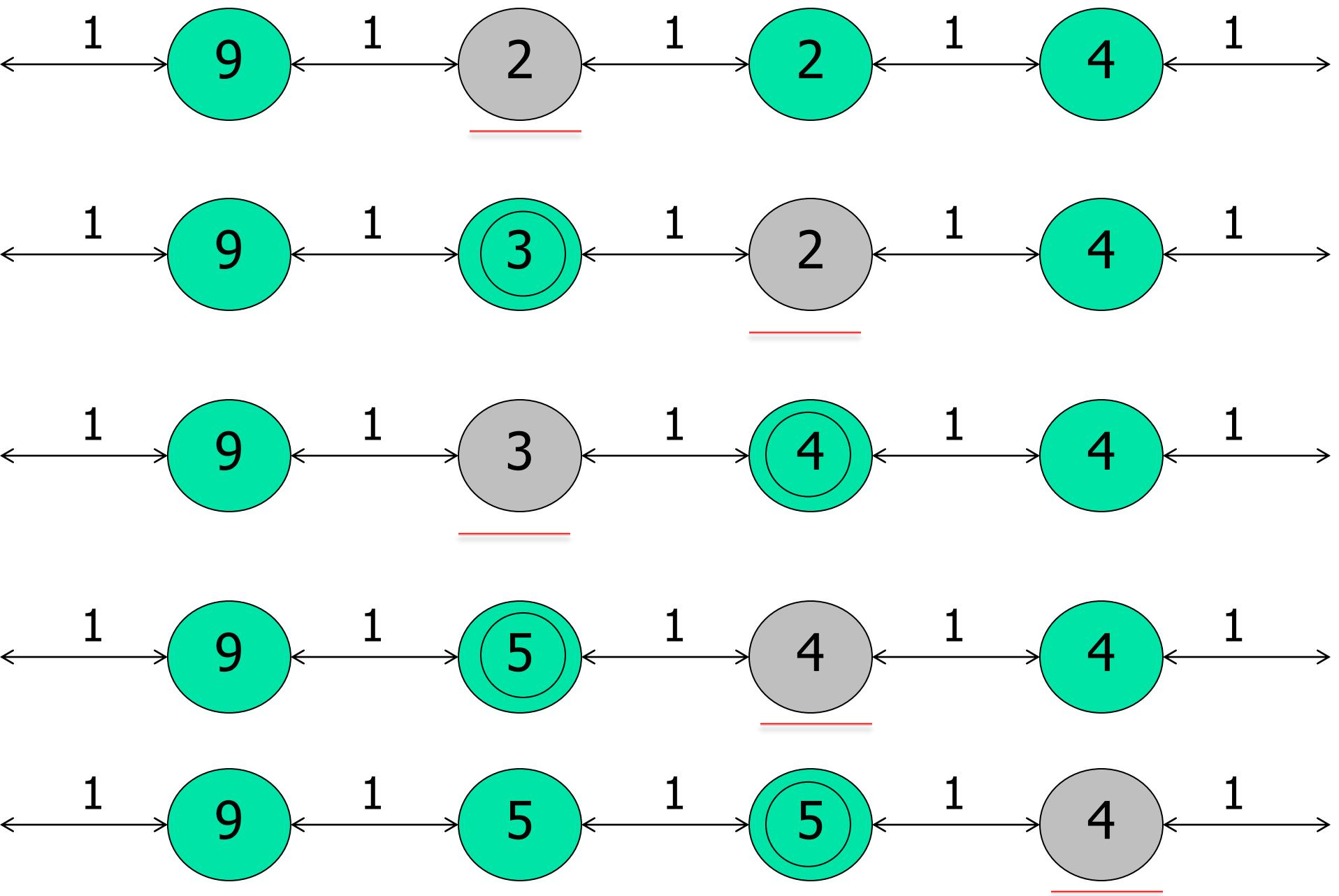
Spatiu de stari mare – nepractic

Foloseste ca fundament pt A* in timp real

- Modific A* pentru a putea fi utilizat on-line
- De ce nu pot folosi direct A*?

2.2 Learning Real-Time A* (LRTA*)

- Intreprinde calculul miscarii urmatoare cu executia miscarilor si alege miscarile intr-un timp constant
- Construieste si actualizeaza o tabela ce contine estimari eursitice ale distantei fiecarei stari la starea scop.
- Actualizeaza estimarea de cost a starii pe care a parasit-o si alege miscarea (aparent) cea mai buna
- Initial, intrarile in aceasta tabela corespund unor evaluari euristice sau 0 si sunt subestimari
- Prin explorarea repetata a spatiului, valorile sunt actualizate pana cand ajung, intr-un final, la valorile corecte



Learning Real-Time A* (LRTA*)

Agentul considera numai nodul curent

Agent in nodul **i**

Agentul inregistreaza distanta estimata pentru un nod

1. Lookahead

- Pentru fiecare vecin **j** a lui **i** calculeaza

$$f(j) = c(i,j) + h(j)$$

h(j) – estimarea caii $j \rightarrow S_f$

c(i,j) – cost arc $i \rightarrow j$

Learning Real-Time A* (LRTA*)

2. Actualizeaza h

- Actualizeaza h estimat a nodului **i**

$$h(i) \leftarrow \min_j f(j)$$

3. Selecteaza actiunea si memoreaza

- Mergi la starea **j** cu valoarea **minima** $f(j)$ – actiunea **a**
- Memoreaza $i \rightarrow_a j$

Valorile initiale ale lui **h** trebuie sa fie admisibile

Learning Real-Time A* (LRTA*)

- Agentul aflat in starea s' face backtrack intr-o stare anterior vizitata s in momentul in care estimarea rezolvării problemei din starea s' + costul de intoarcere in s este *mai mic* decat costul de a merge inainte din starea s' .
- In LRTA* meritul fiecarui nod este calculat relativ la pozitia curenta (nu la starea initiala).
- Este suficient sa memoram *tabela hash* H cu h a nodurilor deja vizitate

Learning Real-Time A* (LRTA*)

Algoritm LRTA*(s') intoarce o actiune

intrari: s' , o perceptie care identifica starea curenta

variabile globale: Rezultat: o tabela indexata dupa stari si actiuni, initial vida

H – o tabela cu estimarea starilor, initial vida

s, a – starea anterioara si actiunea anterioara, initial nule

daca s' este stare scop **atunci intoarce** stop

daca s' este stare noua ($s' \notin H$) **atunci** $H[s'] \leftarrow h(s')$

daca $s \neq \text{null}$ **atunci**

 Rezultat[s,a] $\leftarrow s'$

$H[s] \leftarrow \min_{b \in \text{Actiuni}(s)} \text{LRTA}^*-\text{Cost}(s, b, \text{Rezultat}[s,b], H)$

$a \leftarrow$ o actiune b din $\text{Actiuni}(s')$ care minimizeaza $\text{LRTA}^*-\text{Cost}(s', b, \text{Rezultat}[s',b], H)$

$s \leftarrow s'$

intoarce a

Agentul selecteaza o actiune in fct de valorile starilor invecinate,
care sunt actualizate pe masura ce agentul exploreaza spatiul

Algoritm LRTA*-Cost(s,a,s', H) intoarce o estimare de cost

daca s' nu este definita **atunci intoarce** $h(s)$

altfel intoarce $c(s,a,s') + H[s']$

Learning Real-Time A* (LRTA*)

- Intr-un spatiu de cautare finit cu costuri pozitive, in care exista o cale de la orice stare S la S_f si se utilizeaza estimari admisibile nenegative, LRTA* este complet (va ajunge in final in starea scop)
- In plus « invata » solutia optima in timp

2.3 Cautare cu tinta mobila (MTS)

- Generalizare a LRTA*
- Starea scop se schimba pe parcursul cautarii
- MTS – generalizare a LRTA*
- MTS trebuie sa obtina informatie despre locatia starii scop
- Consideram spatiul ca un graf neorientat
- PS nu are harta mediului dar stie pozitia T
- PS stie starile adiacente

Cautare cu tinta mobila

- Presupunem ca PS si T se misca alternativ
- Presupunem ca viteza T este mai mica decat cea a PS (la cativa pasi T sta pe loc)
- Stare finala (scop) = PS si T au aceeasi pozitie
- x, x' – pozitia curenta si pozitia vecinilor pt PS
- y, y' – pozitia curenta si pozitia urmatoare pt T

Cautare cu tinta mobila (MTS)

- Presupun ca toate arcele au ***cost 1***
- Matrice de valori $h(x,y)$ – estimarea distantei intre PS si T
- Matricea poate fi mare dar este suficient sa actualizam numai valorile care se schimba
- h admisibila
- 2 evenimente, fiecare face o actualizare a valorii euristicilor:
 - Miscare a PS
 - Miscare a T

Cautare cu tinta mobila (MTS)

Miscare PS

- Calculeaza $h(x',y)$ pentru fiecare vecin x' a lui x
- Miscare la x' cu $h(x',y)$ minim

$$x \leftarrow x'$$

- Actualizeaza valoarea $h(x,y)$ astfel

$$h(x,y) = \max \{ h(x,y), \min_{x'} (h(x',y)+1) \}$$

Deoarece orice cale de la x la y trebuie sa treaca prin vecinul x' , costul drumului minim de la x la y va fi cel putin la fel de mare ca cel al drumului prin oricare vecin al lui x .

Cautare cu tinta mobila (MTS)

Miscare T

PS observa miscarea T de la y la y'

- Calculeaza $h(x, y')$ pentru noua pozitie y' a lui T
- Actualizeaza valoarea $h(x, y)$ astfel

$$h(x, y) = \max \{ h(x, y), h(x, y') - 1 \}$$

- Actualizeaza starea scop cu noua pozitie a lui T

$$y \leftarrow y'$$

Daca $h(x, y') > h(x, y) + 1$ atunci $h(x, y) \leftarrow h(x, y') - 1$
deoarece, distanta y si y' fiind 1, distanta la vechea
pozitie trebuie sa fie cel putin la fel de mare ca
distanta la noua pozitie -1

Cautare cu tinta mobila (MTS)

- Intr-un spatiu de cautare finit cu costuri pozitive in care exista o cale de la fiecare stare S la starea scop S_f , daca se porneste cu valori ale functiei euristice admisibile si se permit miscari ale PS si T in orice directie cu cost unitar, PS care executa MTS va ajunge la T daca T sare periodic peste miscari.

3. Problema satisfacerii restrictiilor

$$\begin{array}{lll} \{X_1, \dots, X_n\} & R_j \subset D_{i1} \times \dots \times D_{ij} & X_{i1}, \dots, X_{ij} \\ \{D_1, \dots, D_n\} & & \\ \{R_1, \dots, R_k\} & & \{(X_1, x_{j_1}), \dots, (X_n, x_{j_n})\} \end{array}$$

- Gradul unei variabile
- Aritatea unei restrictii
- Gradul problemei
- Aritatea problemei

Instante ale CSP

- Determinarea unei solutii sau a tuturor solutiilor
 - CSP totala
 - CSP partiala
 - CSP binara – graf de restrictii
- CSP – problema de cautare, in NP
- Sub-clase de probleme cu timp polinomial
 - Reducerea timpului (reducerea sp. de cautare)

Notatii

- X_1, \dots, X_N variabilele problemei, N fiind numarul de variabile ale problemei,
- U - intreg care reprezinta indicele variabilei curent selectate pentru a i se atribui o valoare
- F - vector indexat dupa indicii variabilelor, in care sunt memorate selectiile de valori facute de la prima variabila si pana la variabila curenta

$$\text{Relatie}(U_1, F[U_1], U_2, F[U_2]) = \begin{cases} \text{adevarat} & \text{daca exista restrictia } R_{U_1 U_2}(F[U_1], F[U_2]) \\ & R_{U_1 U_2} \subset D_{U_1} \times D_{U_2} \\ \text{fals} & \text{in caz contrar} \end{cases}$$

3.1 Im bunatatirea performantelor BKT

1) Algoritmi care modifica spatiul de cautare prin eliminarea unor regiuni ce nu contin solutii

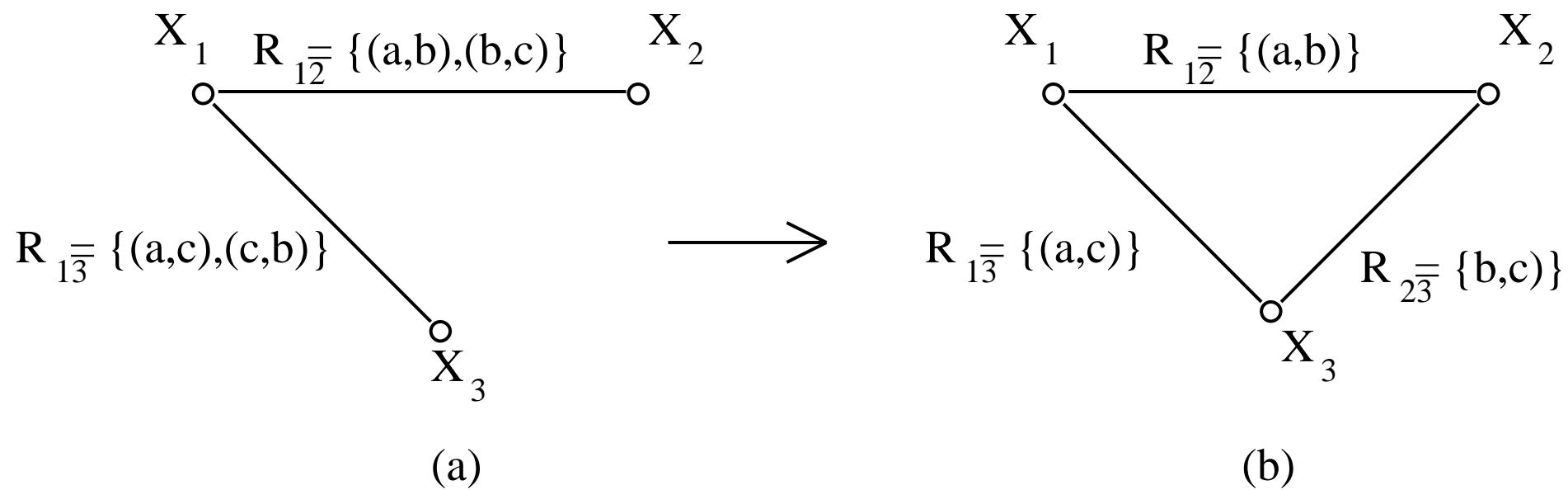
- Algoritmi de imbunatatire a consistentei reprezentarii (inainte de cautare)
 - Consistenta locala a arcelor sau a cailor in graful de restrictii
- Algoritmi hibrizi (in timpul cautarii)
 - Im bunatatesc performantele rezolvării prin reducerea numărului de teste.
 - *Tehnici prospective:*
 - Algoritmul de cautare cu predictie completa
 - Algoritmul de cautare cu predictie parțiala
 - Algoritmul de cautare cu verificare predictiva
 - *Tehnici retrospective:*
 - Algoritmul de backtracking cu salt
 - Algoritmul de backtracking cu marcare
 - Algoritmul de backtracking cu multime conflictuala

2) Utilizarea euristicilor

Propagarea locală a restrictiilor

■ Propagarea restrictiilor

$$D_{X_1} = D_{X_2} = D_{X_3} = \{a, b, c\}$$



Propagarea locala a restrictiilor

- Combinatia de valori x si y pentru variabilele X_i si X_j este permisa de restrictia explicita $R_{ij}(x,y)$.
- Un **arc** (X_i, X_j) intr-un graf de restrictii orientat se numeste ***arc-consistent*** daca si numai daca pentru orice valoare $x \in D_i$, domeniul variabilei X_i , exista o valoare $y \in D_j$, domeniul variabilei X_j , astfel incat $R_{ij}(x,y)$.
- **Graf de restrictii orientat *arc-consistent***

Algoritm: AC-3: Realizarea arc-consistentei pentru un graf de restrictii.

1. Creeaza o coada $Q \leftarrow \{ (X_i, X_j) \mid (X_i, X_j) \in \text{Multime arce}, i \neq j \}$
2. **cat timp** Q nu este vida **executa**
 - 2.1. Elimina din Q un arc (X_k, X_m)
 - 2.2. Verifica(X_k, X_m)
 - 2.3. **daca** subprogramul Verifica a facut schimbari in domeniul variabilei X_k
atunci

$$Q \leftarrow Q \cup \{ (X_i, X_k) \mid (X_i, X_k) \in \text{Multime arce}, i \neq k, m \}$$

sfarsit.

Verifica (X_k, X_m)

pentru fiecare $x \in D_k$ **executa**

1. **daca** nu exista nici o valoare $y \in D_m$ astfel incat $R_{km}(x, y)$
atunci elibima x din D_k

sfarsit.

Cale-consistenta

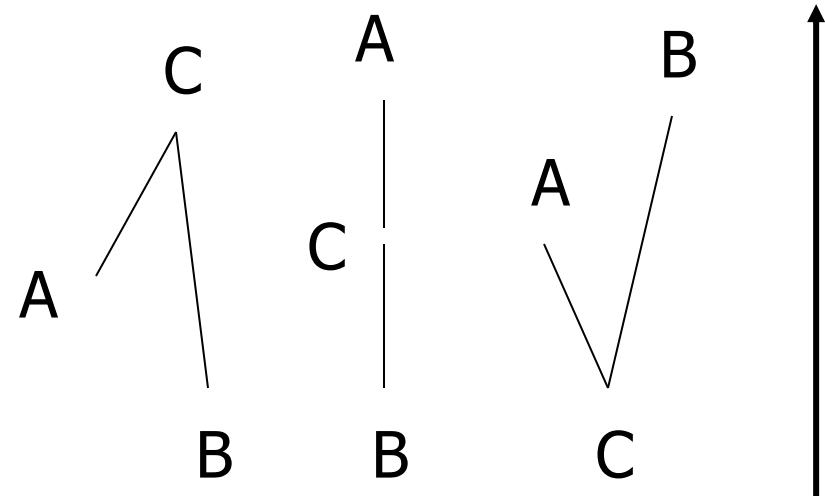
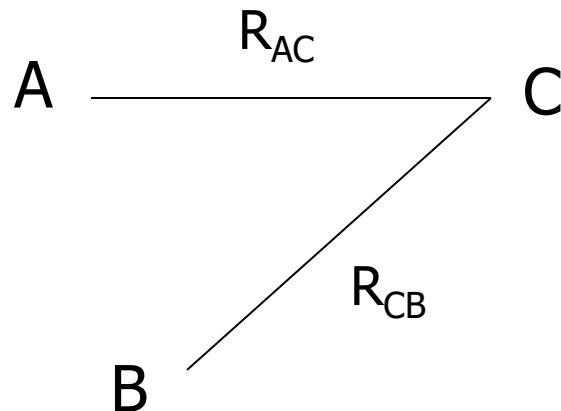
- O cale de lungime m prin nodurile i_0, \dots, i_m ale unui graf de restrictii orientat se numeste ***m-cale-consistenta*** daca si numai daca pentru orice valoare $x \in D_{i_0}$, domeniul variabilei i_0 si o valoare $y \in D_{j_m}$, domeniul variabilei i_m , pentru care $R_{i_0 i_m}(x, y)$, exista o secventa de valori $z_1 \in D_{i_1} \dots z_{m-1} \in D_{i_{m-1}}$ astfel incat $R_{i_0 i_1}(x, z_1), \dots, R_{i_{m-1} i_m}(z_{m-1}, y)$
- **Graf de restrictii orientat *m-arc-consistent***
- n-cale-consistenta
- 2-cale consistenta

Complexitate

- N - numarul de variabile
- a - cardinalitatea maxima a domeniilor de valori ale variabilelor
- e - numarul de restrictii.
- Algoritmului de realizare a arc-consistentei - *AC-3*: complexitate timp este $O(e*a^3)$; complexitate spatiu: $O(e+N*a)$
- S-a gasit si un algoritm de complexitate timp $O(e*a^2)$ – *AC-4*
- Algoritmul de realizare a 2-cale-consistentei - *PC-4*: complexitatea timp $O(N^3*a^3)$

CSP fara bkt - conditii

- Graf de restrictii ordonat
- Latimea unui nod
- Latimea unei ordonari a nodurilor
- Latimea unui graf de restrictii



Teoreme

- Daca un graf de restrictii **arc-consistent** are *latimea egala cu unu* (i.e. este un arbore), atunci problema asociata grafului admite o solutie fara backtracking.
- Daca un graf de restrictii **2-cale-consistent** are *latimea egala cu doi*, atunci problema asociata grafului admite o solutie fara backtracking.

d-consistenta

- Fiind data o ordonare d a variabilelor unui graf de restrictii R , **graful R este d -arc-consistent** daca toate arcele avand directia d sunt arc-consistente.
- Fie un graf de restrictii R , avind ordonarea variabilelor d cu latimea egala cu unu. Daca R este d -arc-consistent atunci cautarea dupa directia d este fara backtracking.

d-consistenta

Algoritm AC-4: Realizarea d-arc-consistentei unui graf de restrictii cu ordonarea variabilelor (X_1, \dots, X_N)

pentru $I \leftarrow N$ la 1 executa

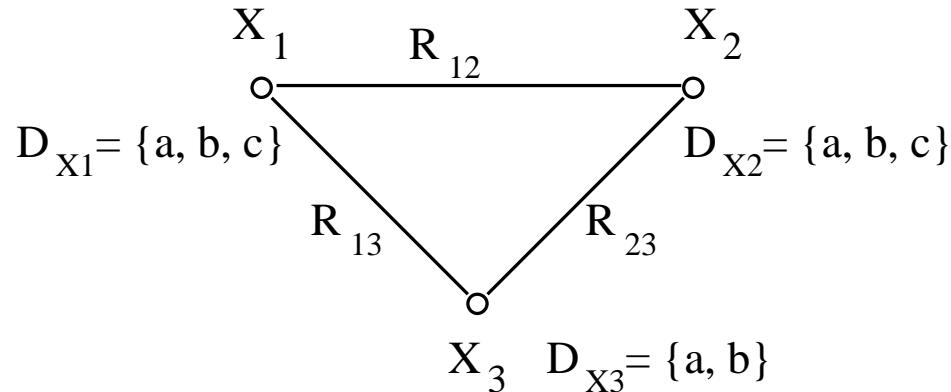
1. pentru fiecare arc (X_j, X_i) cu $j < i$ executa

1.1. Verifica(X_j, X_i)

sfarsit.

- Complexitatea timp: $O(e * a^2)$

Cale-consistenta – reprezentare matriceala



$$R_{11} = \begin{matrix} a & \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ b & \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \\ c & \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \end{matrix} \quad R_{12} = \begin{matrix} a & \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \\ b & \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \\ c & \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \end{matrix} \quad R_{13} = \begin{matrix} a & \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \\ b & \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \\ c & \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \end{matrix}$$

$$R_{22} = \begin{matrix} a & \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ b & \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \\ c & \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \end{matrix} \quad R_{23} = \begin{matrix} a & \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \\ b & \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \\ c & \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \end{matrix} \quad R_{33} = \begin{matrix} a & \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ b & \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \\ c & \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix} \end{matrix}$$

Operatii asupra matricilor de restrictii

- **Intersectia** a doua restrictii - o restrictie care permite numai perechile de valori ce apar în ambele restrictii.
- Intersectia a doua restrictii se notează cu &.
- **Componerea** a doua restrictii, R_{12} și R_{23} , între variabilele X_1 și X_2 și respectiv X_2 și X_3 , determină o **nouă restrictie R_{13}** definită astfel: $(X_1, X_3) \in R_{13}$ dacă există $x_2 \in D_{X2}$ astfel încât $(X_1, X_2) \in R_{12}$ și $(X_2, X_3) \in R_{23}$
- În notatia cu matrice, compunerea a două restrictii se poate obține prin produsul de matrice $R_{13} = R_{12} * R_{23}$ și înlocuind în matricea rezultat valorile pozitive cu 1.

Algoritm: PC4- Realizarea d-cale-consistentei unui graf de restrictii cu variabile ordonate

$d = (X_1, X_2, \dots, X_n) \quad (X_1, X_2, \dots, X_n)$

V – noduri, E - arce

1. $Y \leftarrow \{R_{ij} \mid 1 \leq i, j \leq N\}$
2. pentru $k \leftarrow N$ pana la 1 execută
 - 2.1. pentru fiecare pereche (i, k) cu $1 \leq i \leq k$ si $(X_i, X_k) \in E$ execută
 - 2.1.1. $Y_{ii} \leftarrow Y_{ii} \& (Y_{ik} \cdot Y_{kk} \cdot Y_{ki})$
 - 2.2. pentru fiecare triplet (i, j, k) cu $1 \leq i, j \leq k$ si $(X_i, X_k) \in E$ si $(X_j, X_k) \in E$ execută
 - 2.2.1. $Y_{ij} \leftarrow Y_{ij} \& (Y_{ik} \cdot Y_{kk} \cdot Y_{kj})$
 - 2.2.2. $E \leftarrow E \cup (X_i, X_j)$

sfarsit.

Tehnici perspective

- Conventii
- Notatii: U, N, F ($F[U]$), T ($T[U] \dots X_U$), TNOU, LINIE_VIDA

$$\text{Relatie}(U_1, L_1, U_2, L_2) = \begin{cases} \text{adevarat} & \text{daca } R_{U_1 U_2}(L_1, L_2) \\ \text{fals} & \text{în caz contrar} \end{cases}$$

- *Verifica_Inainte*
- *Verifica_Viitoare*
- **Predictie completa**
- **Predictie parțială**
- **Verificare predictiva**

Algoritm: Backtracking cu predictie completa

Predictie(U, F, T)

pentru fiecare element L din T[U] **executa**

1. $F[U] \leftarrow L$
2. **daca** $U < N$ **atunci** *//verifica consistenta atribuirii*
 - 2.1 $TNOU \leftarrow \text{Verifica_Inainte}(U, F[U], T)$
 - 2.2 **daca** $TNOU \neq \text{LINIE_VIDA}$
atunci $TNOU \leftarrow \text{Verifica_Viitoare}(U, TNOU)$
 - 2.3 **daca** $TNOU \neq \text{LINIE_VIDA}$
atunci **Predictie** ($U+1, F, TNOU$)
3. **altfel** afiseaza atribuirile din F

sfarsit

Verifica_Inainte (U, L, T)

1. $TNOU \leftarrow$ tabela vida
2. **pentru** $U2 \leftarrow U+1$ pana la N **executa**
 - 2.1 **pentru** fiecare element $L2$ din $T[U2]$ **executa**
 - 2.1.1 daca $\text{Relatie}(U, L, U2, L2) = \text{adevarat}$
atunci introduce $L2$ in $TNOU[U2]$
 - 2.2 **daca** $TNOU[U2]$ este vida
atunci intoarce LINIE_VIDA
3. **intoarce** $TNOU$
sfarsit

Verifica_Viitoare (U, TNOU)

daca $U+1 < N$ **atunci**

1. pentru $U1 \leftarrow U+1$ pana la N **executa**

1.1 pentru fiecare element L1 din TNOU[U1] **executa**

1.1.1 pentru $U2 \leftarrow U+1$ pana la N, $U2 \neq U1$ **executa**

i. pentru fiecare element L2 din TNOU[U2] **executa**

- **daca** Relatie (U1, L1, U2, L2) = adevarat
atunci intrerupe ciclul //dupa L2

ii. daca nu s-a gasit o valoare consistenta pentru U2
atunci

- elimina L1 din TNOU[U1]
- **intrerupe ciclul** // dupa U2

1.2 daca TNOU[U1] este vida

atunci intoarce LINIE_VIDA

2. intoarce TNOU

sfarsit

BKT cu predictie parțială

- Se modifica Verifica_Viitoare pasii marcati cu rosu

Verifica_Viitoare (U, TNOU)

daca $U+1 < N$ atunci

1. **pentru** $U1 \leftarrow U+1$ pana la $N - 1$ **executa**

 1.1 **pentru** fiecare element $L1$ din $TNOU[U1]$ **executa**

 1.1.1 **pentru** $U2 \leftarrow U1+1$ pana la N **executa**

 i. **pentru** fiecare element $L2$ din $TNOU[U2]$ **executa**

 - **daca** Relatie ($U1, L1, U2, L2$) = adevarat
 atunci intrerupe ciclul //dupa $L2$

 ii. **daca** nu s-a gasit o valoare consistentă pentru $U2$
 atunci

 - elimina $L1$ din $TNOU[U1]$
 - **intrerupe ciclul** // dupa $U2$

 1.2 **daca** $TNOU[U1]$ este vida

atunci intoarce LINIE_VIDA

2. **intoarce** TNOU

sfarsit

BKT cu verificare predictiva

- Se elimina apelul Verifica_Viitoare(U, TNOU) in subprogramul Predictie

Algoritm: **Backtracking cu verificare predictiva**

Predictie(U, F, T)

pentru fiecare element L din T[U] **executa**

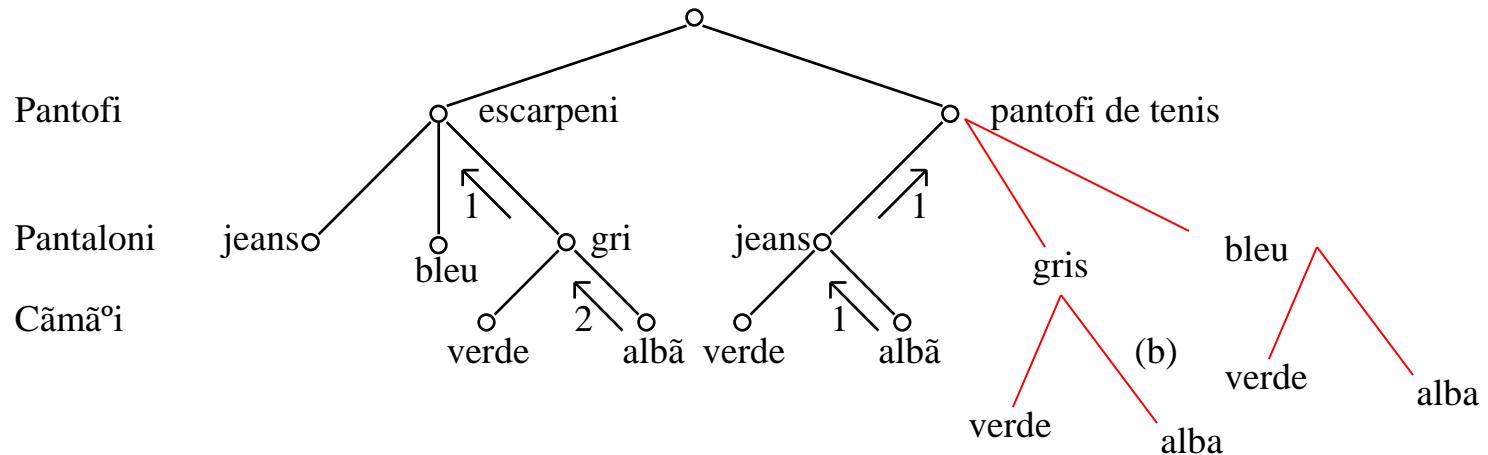
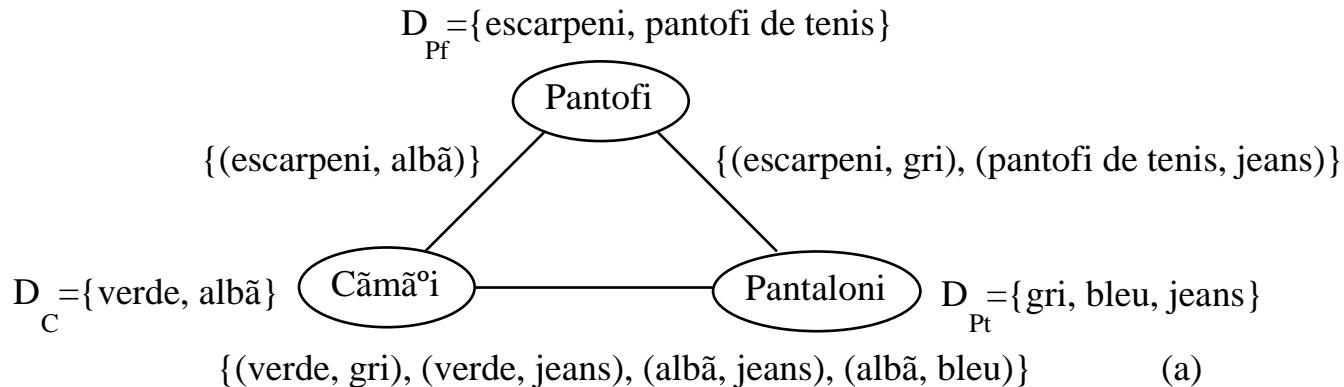
1. $F[U] \leftarrow L$
2. **daca** $U < N$ **atunci** //verifica consistenta atribuirii
 - 2.1 $TNOU \leftarrow \text{Verifica_Inainte}(U, F[U], T)$
 - 2.2 **daca** $TNOU \neq \text{LINIE_VIDA}$

~~atunci $TNOU \leftarrow \text{Verifica_Viitoare}(U, TNOU)$~~
2.2 **daca** $TNOU \neq \text{LINIE_VIDA}$
atunci Predictie ($U+1, F, TNOU$)
3. **altfel** afiseaza atribuirile din F

sfarsit

Tehnici retrospective

■ Backtracking cu salt



Algoritm: Backtracking cu salt

BacktrackingCuSalt(U, F) /* intoarce Nivel – blocare */

/* NrBlocari, NivelVec, I, test – var locale */

1. NrBlocari $\leftarrow 0$, I $\leftarrow 0$, Nivel $\leftarrow U$

2 **pentru** fiecare element V a lui X_U **executa**

 2.1 $F[U] \leftarrow V$

 2.2 test, NivelVec[I] $\leftarrow \text{Verifica1}(U, F)$

 2.3 **daca** test = adevarat **atunci**

 2.3.1 **daca** $U < N$ **atunci**

 i. Nivel $\leftarrow \text{BacktrackingCuSalt}(U+1, F)$

 ii. **daca** Nivel $< U$ **atunci** salt la pasul 4

 2.3.2 **altfel** afiseaza valorile vectorului F // solutia

 2.4 **altfel** NrBlocari $\leftarrow \text{NrBlocari} + 1$

 2.5 I $\leftarrow I + 1$

3. **daca** NrBlocari = numar valori ale lui $X[U]$ si
toate elementele din NivelVec sunt egale
atunci Nivel $\leftarrow \text{NivelVec}[1]$

4. **intoarce** Nivel

sfarsit

Verifica1 (U, F)

/* intoarce test si nivelul la care s-a produs blocarea sau 0 */

1. test \leftarrow adevarat

2. I \leftarrow U-1

3. **cat timp** I>0 **executa**

 3.1 test \leftarrow Relatie(I, F[I], U, F[U])

 3.2 **daca** test = fals

atunci intrerupe ciclul

 3.3 I \leftarrow I - 1

4. NivelAflat \leftarrow I

5. **intoarce** test, NivelAflat

sfarsit

Euristici

Utilizarea euristicilor în rezolvarea problemei satisfacerii restrictiilor se realizează prin considerarea următoarelor ordonări: ordonarea variabilelor în vederea atribuirii, ordonarea valorilor în vederea atribuirii, ordonarea testelor variabilelor în rezolvare.

- O combinație adecvată a acestor patru criterii permite implementarea unor euristic simple, cu un cost computational scăzut.

Există două criterii de bază pentru ghidarea căutării.

- Dacă se cere gasirea tuturor soluțiilor problemei, se folosesc criteriul primei blocări, în care sunt încercate întai cele mai probabile cai care ar putea conduce la insucces. Deoarece în cazul determinării tuturor soluțiilor, tot spațiul de căutare trebuie parcurs, detectarea cat mai devreme a cailor care nu duc la soluție prezintă avantajul reducerii spațiului de căutare.
- Dacă se cere gasirea unei singure soluții, este bun criteriul caii promitatoare, i.e. criteriul invers. Se încearcă întai caile cele mai promitatoare pentru gasirea unei soluții.

Euristici

Ordonarea variabilelor

- Aceasta euristică se bazează pe alegerea unei ordonări a variabilelor astfel încât variabilele legate prin restrictii explicite să fie consecutive. Acest criteriu încearcă să evite, pe cat posibil, atribuirile de valori între variabile legate prin restrictii implicite. Restrictiile explicite sunt cele specificate de multimea de restrictii definită în problema. Ele se numesc astăzi pentru a le deosebi de restrictiile implicite, care apar ca efect lateral al celor explicite în procesul de inspectare a spațiului de căutare.
- Dacă se cer toate soluțiile, variabilele care apar într-un număr mic de restrictii și au domenii de valori cu cardinalitate mică, sunt preferate, deci așezate la început în ordonare.
- Dacă se cere numai o soluție, criteriul opus celui enunțat mai sus este cel adecvat. În acest caz, se încearcă mai întâi atribuirile de valori pentru variabilele cu domenii de valori cu cardinalitate mare și legate printr-un număr mare de restrictii.

Euristici

Ordonarea valorilor

- Aceasta euristica tine cont de faptul ca nu toate valorile din domeniul variabilelor apar în toate restrictiile. Dacă se cer toate soluțiile, pentru o variabilă fixată se alege ca prima valoare de atribuit variabilei cea mai restrictionată valoare, i.e. cea care permite cele mai puține atribuiri. Dacă se cere gasirea unei singure soluții, se utilizează criteriul contrar.

Ordonarea testelor

- În tehniciile retrospective, după atribuirea unei valori unei variabile, se testează valoarea variabilei curente cu toate valorile atribuite variabilelor deja considerate. O euristica utilă este aceea de a începe cu variabila precedenta cea mai restrictionată.
- În tehniciile prospective, se va începe testarea cu variabilele care au domeniile de valori cele mai restrânse (cu cardinalitate mică).

3.2 CSP parțială

- Memoreaza cea mai buna solutie gasita pana la un anumit moment (gen IDA*) – *distanta d* fata de solutia perfecta
- Abandoneaza calea de cautare curenta in momentul in care se constata ca acea cale de cautare nu poate duce la o solutie mai buna
- NI - numarul de inconsistente gasite in "cea mai buna solutie" depistata pana la un moment dat – *limita necesara*

CSP partiala

- S - *limita suficientă* - specifică faptul că o soluție care violează un număr de S restricții (sau mai puține), este acceptabilă.
- PBKT(Cale, Distanță, Variabilă, Valori)
 - Semnificativ argumente
 - Rezultat: GATA sau CONTINUA
- variabile globale: CeaMaiBuna, NI, S

Algoritm: CSP Partiala

PBKT(Cale, Distanta, Variabile, Valori)

/ intoarce GATA sau CONTINUA */*

1. **daca** Variabile = {}

atunci

 1.1 CeaMaiBuna \leftarrow Cale

 1.2 NI \leftarrow Distanta

 1.3 **daca** NI \leq S **atunci** **intoarce** GATA
 altfel **intoarce** CONTINUA

2. **altfel**

 2.1 **daca** Valori = {} **atunci** CONTINUA

/ s-au incercat toate valorile si se revine la var ant. */*

 2.2 **altfel**

 2.2.1 **daca** Distanta \geq NI

atunci **intoarce** CONTINUA

/ revine la var ant pentru gasirea unei solutii mai bune */*

2.2.2 altfel

- i. Var \leftarrow first(Variabile)
- ii. Val \leftarrow first(Valori)
- iii. DistNoua \leftarrow Distanță
- iv. Cale1 \leftarrow Cale
- v. **cat timp** Cale1 $\neq \{\}$ **și** DistNoua $<$ NI **executa**

- (VarC, ValC) \leftarrow first(Cale1)
- **daca** Rel(Var, Val, VarC, ValC) = fals
- **atunci** DistNoua \leftarrow DistNoua + 1
- Cale1 \leftarrow Rest(Cale1)

- vi. **daca** DistNoua $<$ NI **și**

PBKT(Cale+(Val, Var), DistNoua, Rest(Variabile), ValoriNoi)
= GATA

/* ValoriNoi - domeniul de valori asociat primei variabile din Rest(Variabile) */

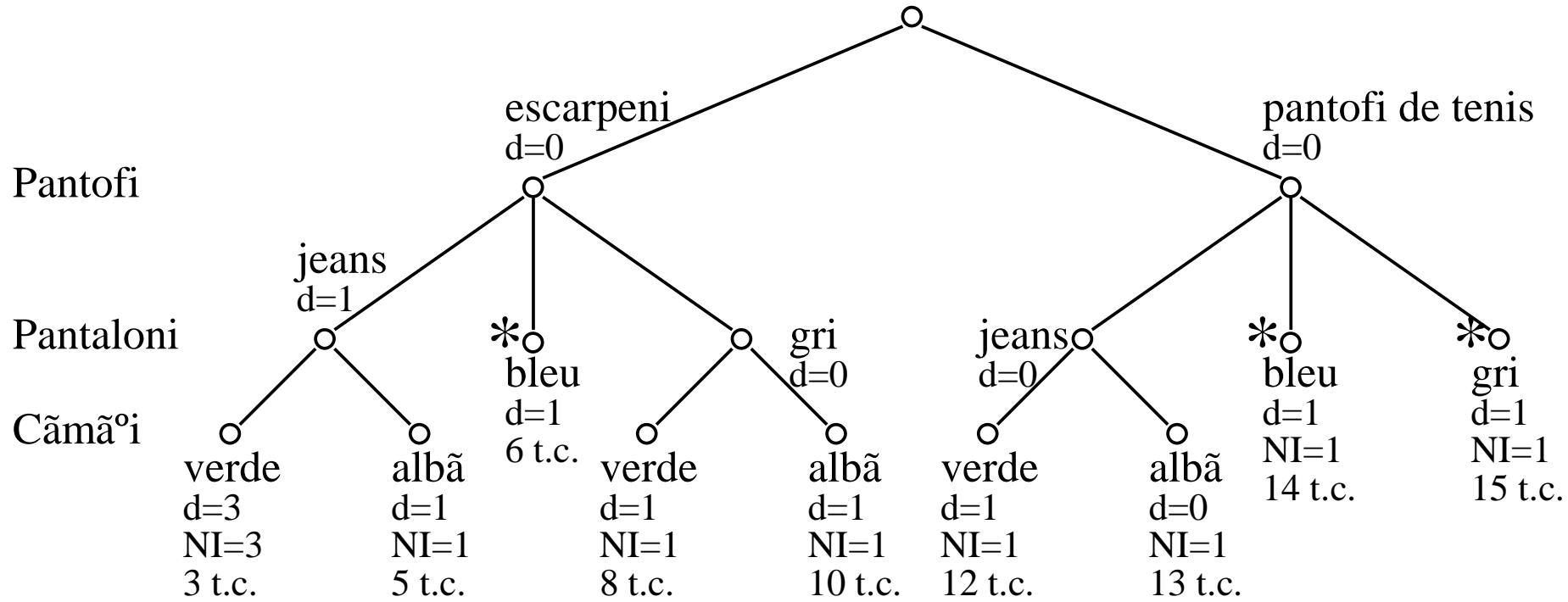
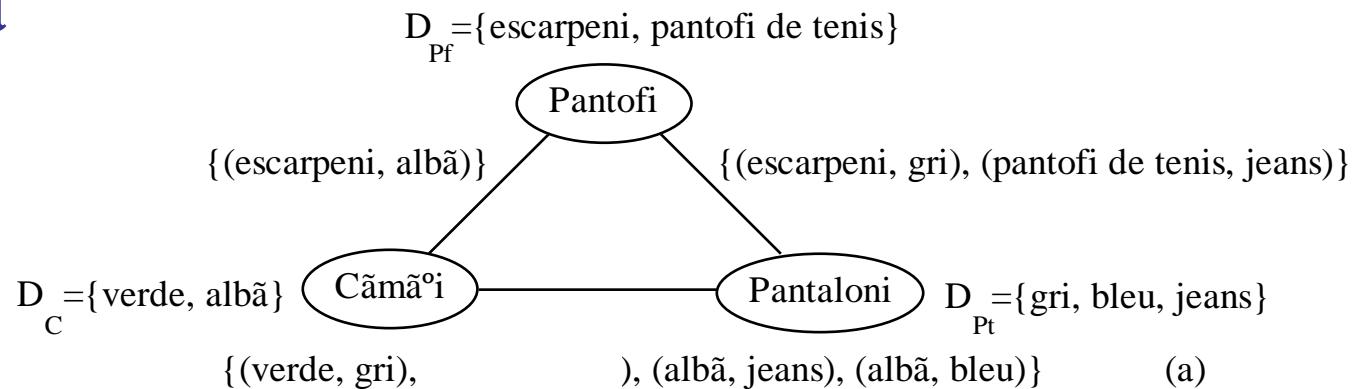
atunci intoarce GATA

altfel intoarce

PBKT(Cale, Distanță, Variabile, Rest(Valori))

sfarsit

CSP partială





Inteligentă Artificială

Universitatea Politehnica Bucuresti
Anul universitar 2020-2021

Adina Magda Florea



Curs 4

Strategii de căutare

- Cautări în jocuri

1. Strategii de căutare în jocuri

- Teoria jocurilor – teoria deciziei pt agenți care interacționează
- Căutări specifice datorita acestei particularități
- S : set de stări cu S_0
- N – număr de jucători
- A – multime de acțiuni
- $f: S \times A \rightarrow S$ (f sau next)
- $Q: S \rightarrow R^N$ – funcția de utilitate / recompensa
- $J: S \rightarrow (1, 2, \dots, N)$ – jucătorul care joaca

1.1 Jocuri cu 2 adversari

- Jocuri ce implică doi adversari ($N=2$)
 - Jucător
 - Adversar
- Algoritmul Minimax
- Algoritmul Alfa-Beta
- Algoritmul Monte Carlo Tree Search

Minimax

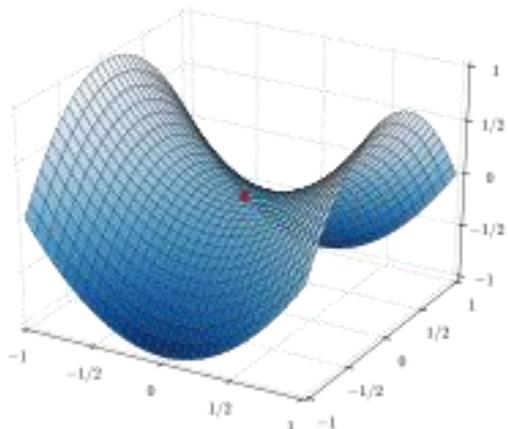
- Jucător – MAX
- Adversar – MIN
- Principiul Minimax
- Etichetez fiecare nivel din AJ cu **MAX** (jucator) și **MIN** (adversar)
- Etichetez frunzele cu scorul jucatorului
- Parcure AJ
 - dacă nodul parinte este **MAX** atunci i se atribuie valoarea maxima a succesorilor sai;
 - dacă nodul parinte este **MIN** atunci i se atribuie valoarea minima a succesorilor sai.

Minimax

- 1944 - von Neumann si Morgenstern descriu cum un proces, numit Minimax, este capabil sa identifice rezultatul final al unui joc si sa aleaga mutarea cea mai buna pentru orice stare de joc, în cazul jocurilor cu informatie perfecta.
- Bazele teoretice ale algoritmului – 1928 - von Neumann

$$f(x,y) = x^2 - y^2$$

$$X \subset \mathbb{R}^n \text{ and } Y \subset \mathbb{R}^m$$



$$f : X \times Y \rightarrow \mathbb{R}$$

$$f(\cdot, y) : X \rightarrow \mathbb{R}$$

$$f(x, \cdot) : Y \rightarrow \mathbb{R}$$

$$\max_{x \in X} \min_{y \in Y} f(x, y) = \min_{y \in Y} \max_{x \in X} f(x, y)$$

Minimax pentru spații de căutare investigate până la o adâncime n

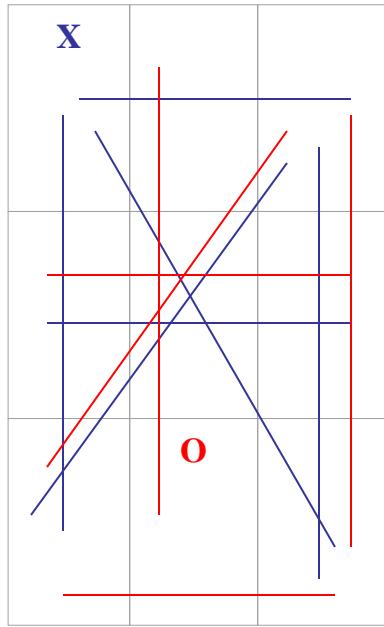
- Spatiul de cautare este f mare sau infinit, nu poate fi investigat exhaustiv
- Algoritmul Minimax pana la o adâncime n
- $nivel(S)$
- O functie euristică de evaluare a unui nod $eval(S)$

Exemplu de funcție de evaluare

Jocul de Tic-Tac-Toe (X si O)

- Functie de estimare euristica $\text{eval}(S)$ - conflictul existent in starea S .
- $\text{eval}(S) = \text{numarul total posibil de linii castigatoare ale lui MAX in starea } S - \text{numarul total posibil de linii castigatoare ale lui MIN in starea } S.$
- Daca S este o stare din care **MAX** poate face o miscare cu care castiga, atunci $\text{eval}(S) = \infty$ (o valoare foarte mare)
- Daca S este o stare din care **MIN** poate castiga cu o singura mutare, atunci $\text{eval}(S) = -\infty$ (o valoare foarte mica).

$\text{eval}(S)$ în Tic-Tac-Toe



X are 6 linii castigatoare posibile

O are 5 linii castigatoare posibile

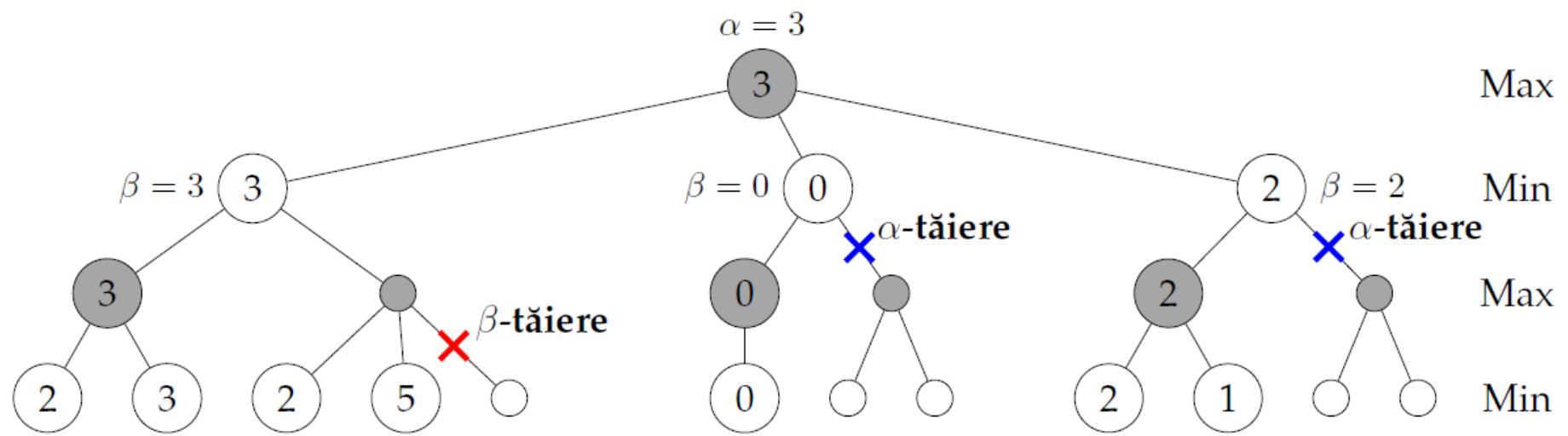
$$\text{eval}(S) = 6 - 5 = 1$$

Algoritmul tăierii alfa-beta

- Este posibil sa se obtină decizia corecta a algoritmului **Minimax** fara a mai inspecta toate nodurile din spatiului de cautare pana la un anumit nivel.
- Procesul de eliminare a unei ramuri din arborele de cautare se numeste *taierea arborelui de cautare (pruning)*.
- Alpha-beta pruning (Knuth and Moore, 1975)

Algoritmul tăierii alfa-beta

- Fie α cea mai buna valoare (cea mai mare) gasita pentru **MAX** si β cea mai buna valoare (cea mai mica) gasita pentru **MIN**.
- Algoritmul **alfa-beta** actualizeaza α si β pe parcursul parcurgerii arborelui si elimina investigarile subarborilor pentru care α sau β sunt mai proaste.
- Terminarea cautarii (taierea unei ramuri) se face dupa doua reguli:
 - **α -taieri** - În cazul în care exista, pentru un nod **Min**, o actiune ce are asociata o valoare $v \leq \alpha$, atunci putem renunta la expandarea subarborelui sau, deoarece **Max** poate atinge deja un castig mai mare, dintr-un subarbore precedent.
 - **β -taieri** - În cazul în care exista, pentru un nod de tip **Max**, o actiune ce are asociata o valoare $v \geq \beta$, atunci putem renunta la expandarea subarborelui sau, deoarece **Min** a limitat deja castigul lui **Max** la β



Algoritm: **Alfa-beta**

MAX(S, α , β) { intoarce valoarea maxima a unei stari. }

0. daca S este nod final **atunci** intoarce scor(S)

1. daca nivel(S) = n **atunci** intoarce eval(S)

2. altfel

2.1 pentru fiecare succesor S_j al lui S **executa**

2.1.1 $\alpha \leftarrow \max(\alpha, \text{MIN}(S_j, \alpha, \beta))$

2.1.2 daca $\alpha \geq \beta$ **atunci** intoarce β

2.2 intoarce α

sfarsit

MIN(S, α , β) { intoarce valoarea minima a unei stari. }

0. daca S este nod final **atunci** intoarce scor(S)

1. daca nivel(S) = n **atunci** intoarce eval(S)

2. altfel

2.1 pentru fiecare succesor S_j al lui S **executa**

2.1.1 $\beta \leftarrow \min(\beta, \text{MAX}(S_j, \alpha, \beta))$

2.1.2 daca $\beta \leq \alpha$ **atunci** intoarce α

2.2 intoarce β

sfarsit

Algoritmul tăierii alfa-beta

- Eficiența algoritmului depinde semnificativ de ordinea de examinare a starilor
- Se recomanda o ordonare euristica a succesorilor, eventual de generat numai primii cei mai buni succesi
- Poate reduce semnificativ timpul de căutare
- *De exemplu – incepe cu cea mai “simplă” miscare sau favorizeaza nodurile cu tăieri B*

Imbunatatiri

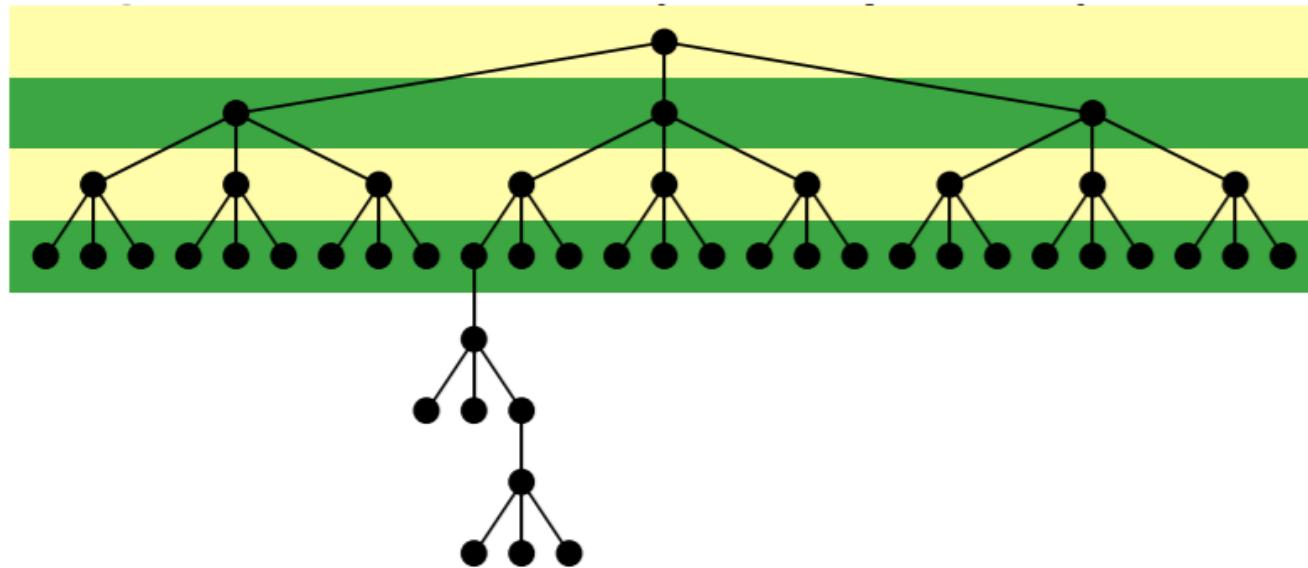
- Timp limitat pentru executarea unei miscari – anytime algorithm
- Foloseste **Iterative Deepening**
- incepe cu ply 1 si obtine cea mai buna miscare
- apoi ply 2 folosind cele mai bune stari cf evaluarii anterioare
- continua cresterea ply pana la expirarea timpului

Imbunatatiri

- Diferite secvente de mutari pot duce la aceleasi pozitii
- Mai multe pozitii de joc pot fi functional echivalente (de ex pozitiile simetrice)
- **Memoize** – tabela hash cu pozitiile de joc pentru a obtine:
 - Estimari ale nodurilor
 - Cea mai buna miscare dintr-un nod

Imbunatatiri

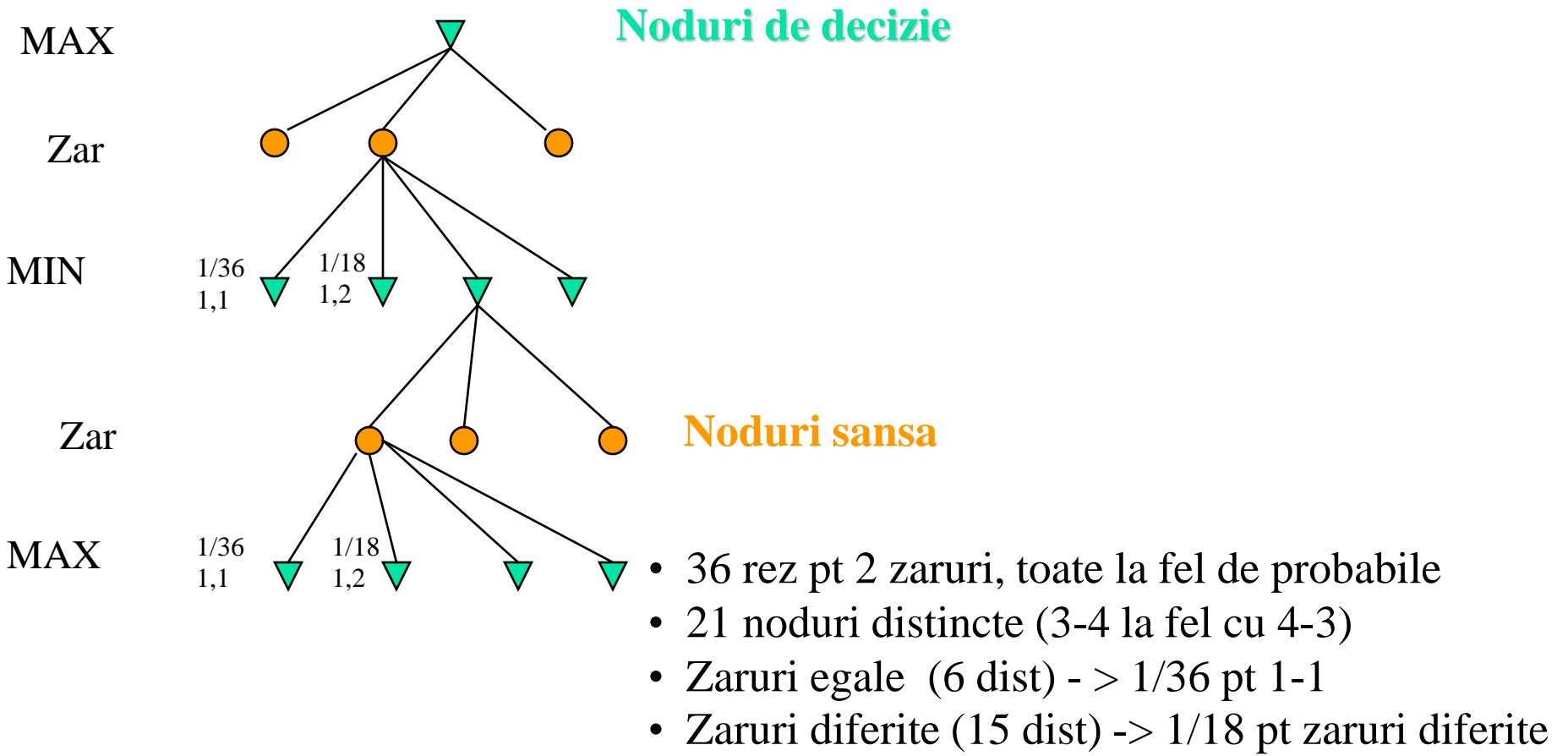
- **Efectul de orizont** – cauta mai mult decat limita de cautare pentru anumite pozitii



Jocuri cu elemente de şansă

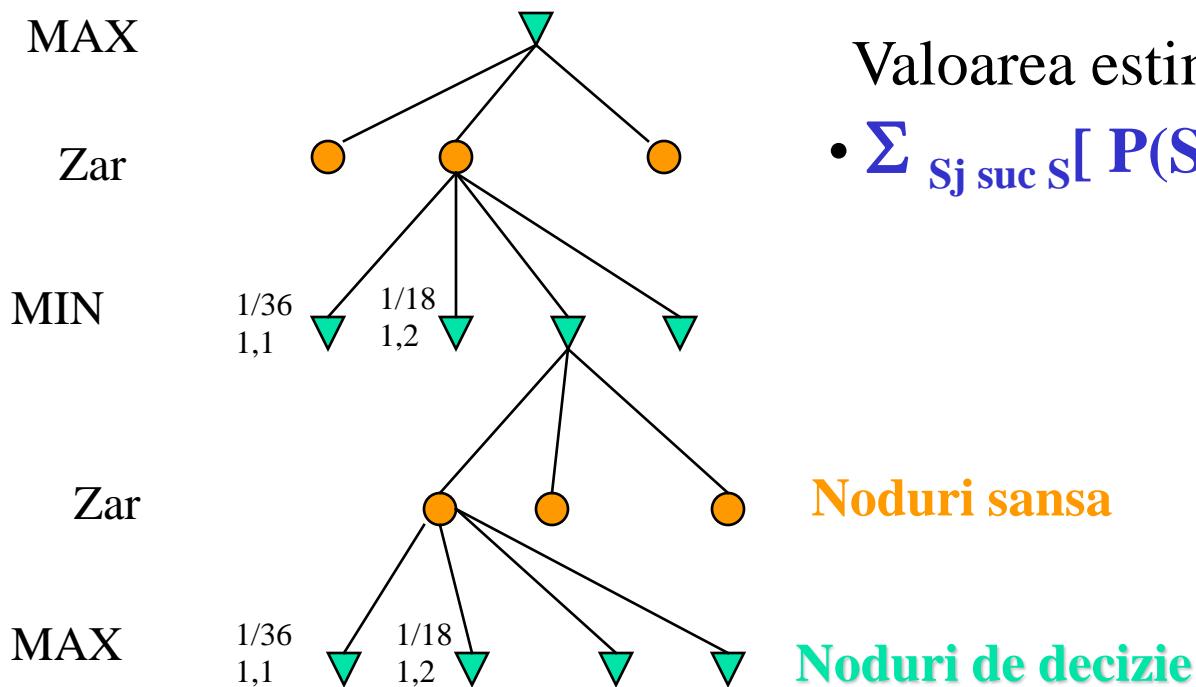
- Jucatorul nu cunoaste mişcările legale ale oponentului
- 3 tipuri de noduri:
 - MAX
 - MIN
 - Şansă (chance nodes)

Noduri şansă – ramurile care pleacă dintr-un nod şansa indică posibile rezultate ale sansei (de exemplu zar)



Functia de evaluare

- scor – nod terminal
- max din Minimax succesorii - MAX
- min din Minimax succesorii - MIN
- $\sum [P(S_j) * \text{Minimax}(S_j)]$ succesorii - SANSA



Valoarea estimata pt noduri sansa

$$\bullet \sum_{S_j \text{ suc } S} [P(S_j) * \text{Minimax}(S_j)]$$

Noduri sansa

Noduri de decizie

1.2 Jocuri cu mai mulți jucători

Exemplu: Chinese checkers (table chinezesti)

- Chinese checkers nu este un joc de origine chineza, ci este o variatie modernă și simplificată, apăruta în Germania prin 1892, a celebrului joc american Halma, inventat de George H. Monks în 1880.
- Abia în momentul în care a devenit cunoscut în Statele Unite ale Americii, a primit denumirea Chinese checkers

George Howard Monks



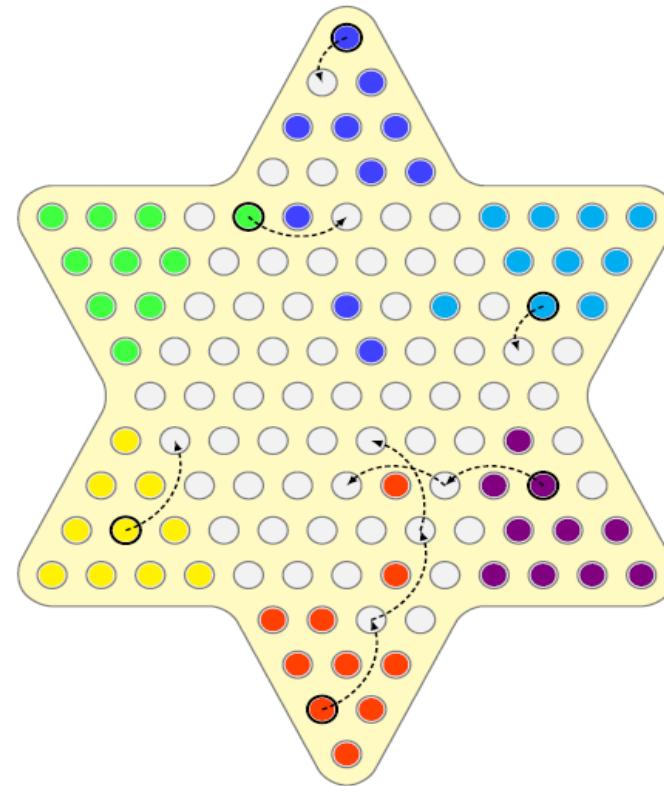
Jocuri cu mai mulți jucători

Chinese checkers

- Este un joc cu informatie perfectă pentru 2-6 jucatori
- Scopul este să deplasam 10 piese dintr-o pozitie de start într-o pozitie finală cat mai repede.



- **Chinese checkers**
- Piezele se muta prin deplasare intr-o pozitie alaturata sau sarind peste pieze alaturate daca exista un loc liber. Se poate sari peste orice numar de pieze si se pot inlantui mai multe sarituri. Piezele nu sunt eliminate dupa sarituri.



Jocuri cu mai mulți jucători

Nu există algoritmi buni consacrați

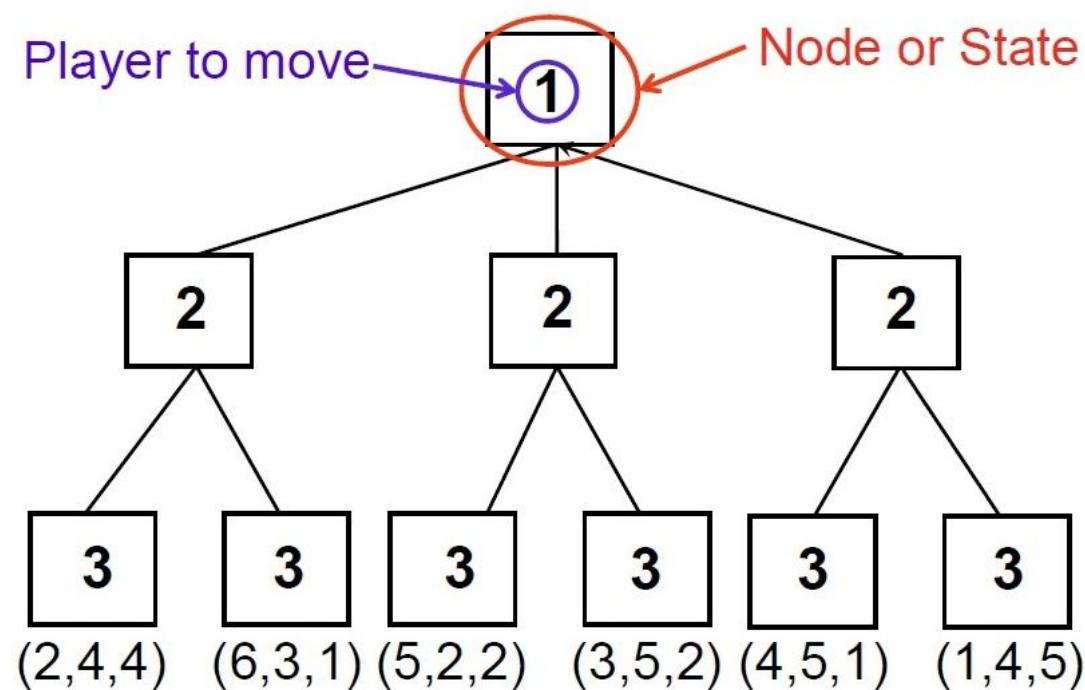
In general, 2 strategii

- **Maxⁿ** - generalizare a Minimax pt **n** jucatori
- **Paranoic** – reduce la joc cu 2 jucatori in care se presupune ca toți ceilalți colaborează împotriva jucatorului simulat

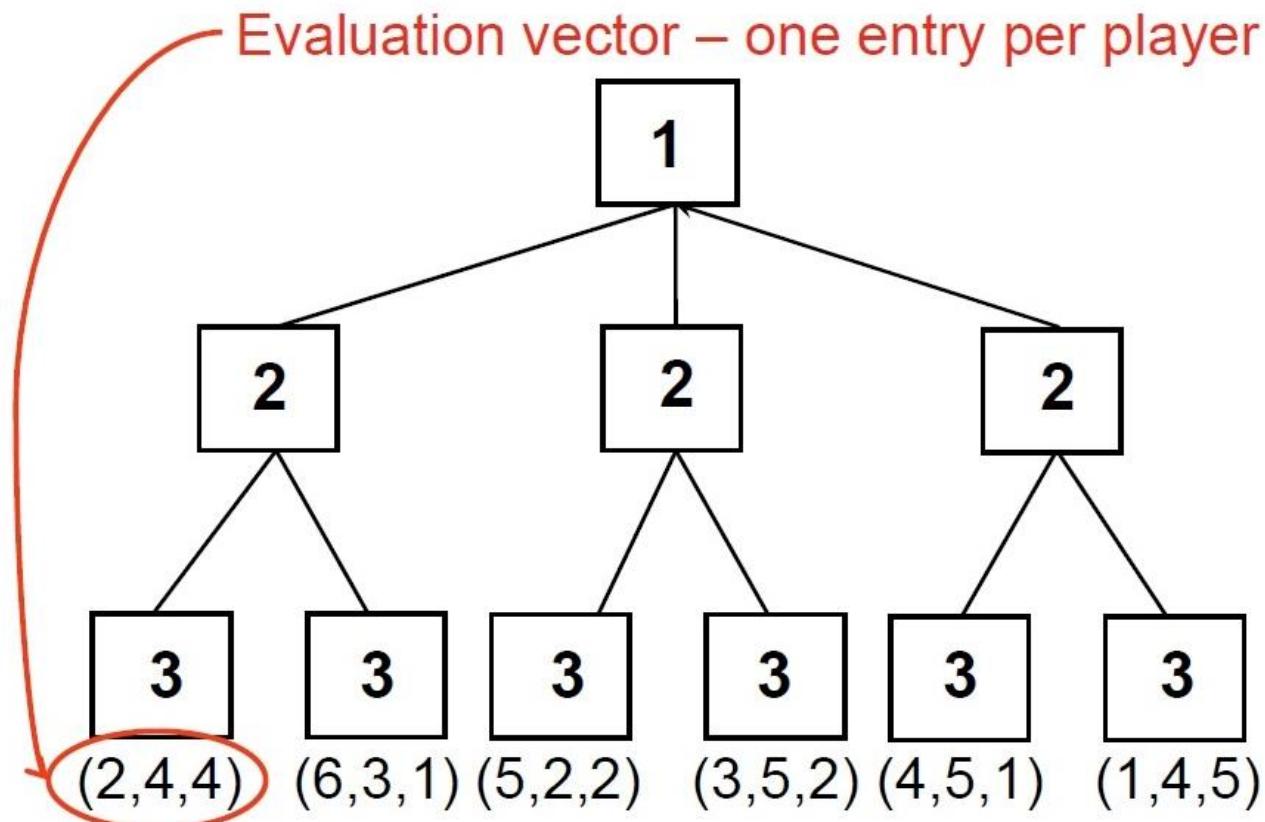
Presupunem jocuri cu informație perfectă

Jocuri cu mai mulți jucători

- Functia de evaluare – dependenta de joc
- Regula de decizie in parcurgerea arborelui de cautare – generica

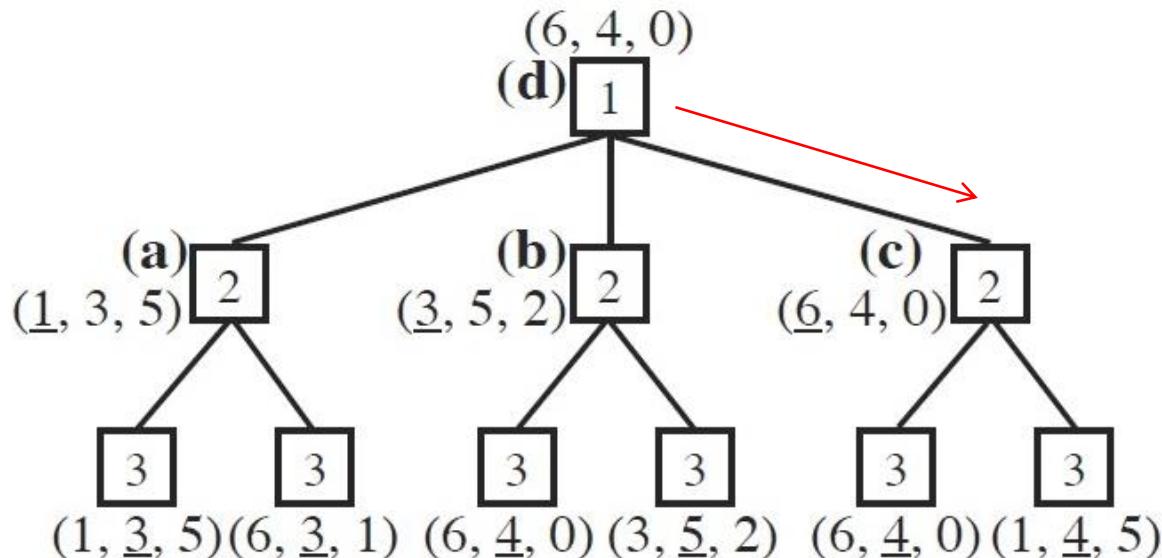


Jocuri cu mai mulți jucători



Strategie Maxⁿ

- Generalizarea Minimax pentru n jucatori
- Frunzele arborelui de joc sunt n-tuple, in care elementul pe pozitia i este scorul jucatorului i .
- Pentru nodurile din interior, valoarea Maxⁿ a unui nod in care jucatorul i muta este valoarea Maxⁿ a successorului pentru care a i -a componenta din vector este maxima.



Strategie Maxⁿ

Maxn(Nod, Juc)

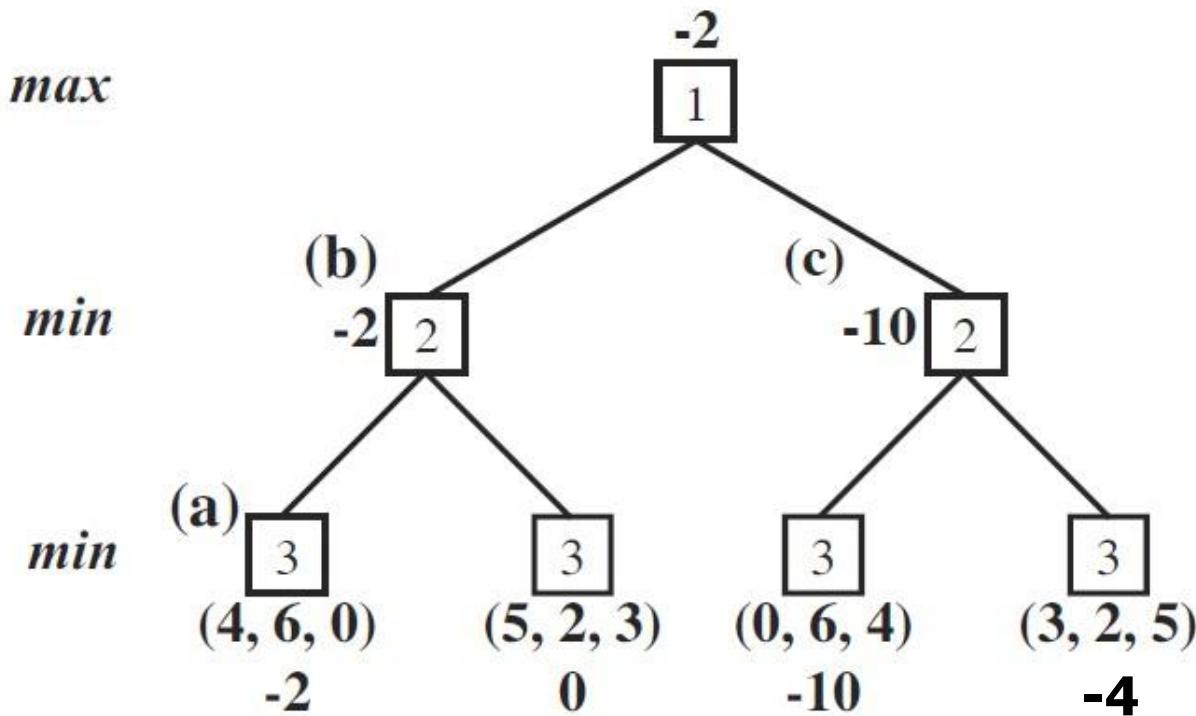
1. daca Nod este nod final **atunci** intoarce scor(Nod[Juc])
2. daca nivel(Nod) = n **atunci** intoarce eval(Nod[Juc])
3. altfel
 - 3.1 P \leftarrow Prim_succesor(Nod)
 - 3.2 Best \leftarrow Maxn (P, Juc_Urm)
 - 3.3 **pentru** fiecare succesor $S_j \neq P$ al lui Nod **executa**
 - Curent \leftarrow Maxn(S_j , Juc_Urm))
 - **if** Curent[Juc] > Best [Juc]
then Best \leftarrow Curent
4. Intoarce Best

Strategie Maxⁿ

- Pot exista multe valori egale Maxⁿ intr-un arbore
 - Rezultatul poate depinde drastic de felul in care se face alegerea
 - E.g., (2,3,3) vs. (2,1,7)
-
- Alpha-beta in adancime nu poate fi aplicat
 - Maxⁿ – shallow pruning, Korf, 1991 (analog to alpha-beta dar cu performante proaste)

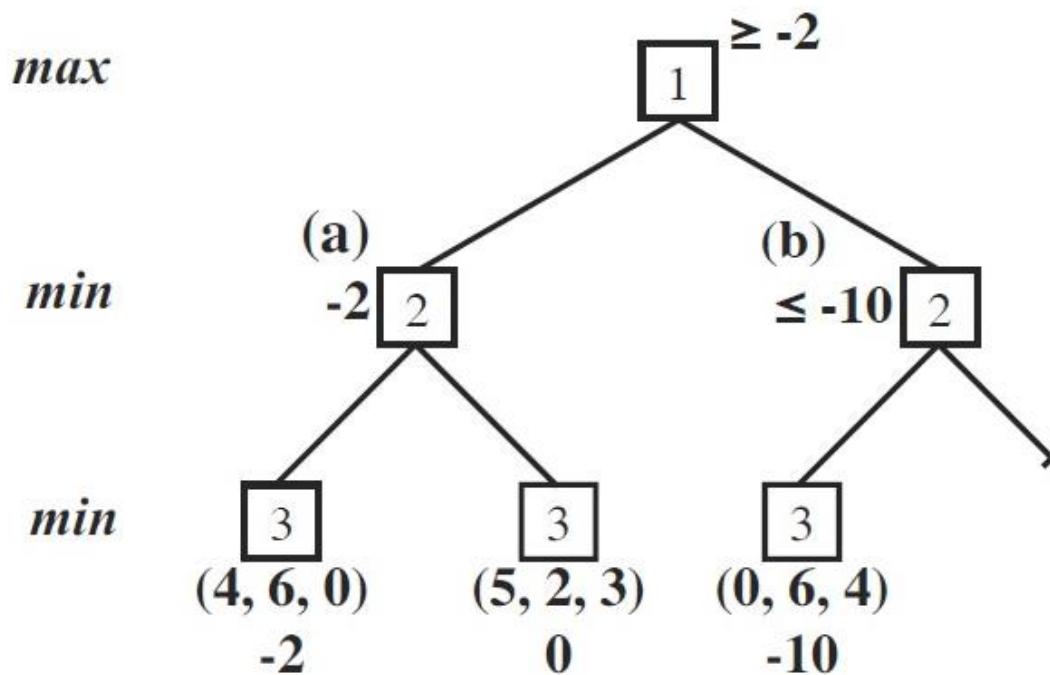
Strategie Paranoic

- **Paranoic** – reduce jocul la 2 jucatori si se poate aplica Minimax



Strategie Paranoic

- La fel ca la Minimax:
 - Exista o unica valoare
 - Se poate utiliza Alpha-Beta



Strategie Paranoic

- Pe masura ce numarul jucatorilor creste beneficiul adus de taiere scade
- Pt jocuri cu 3-6 jucatori, adancimea este cu 20-50% mai mare decat la Maxⁿ
- Presupunerea Paranoic este foarte pesimista
- Exista cazuri in care greseste; in aceasta situatie, cu cat se cauta mai adanc cu atat este mai proasta estimarea

1.3 Monte Carlo Tree Search

- MCTS - algoritm probabilistic care utilizeaza o serie de simulari aleatoare pentru a expanda selectiv arborele de joc
- Reprezinta o metoda buna pentru luarea decizilor în probleme cu un spatiu de cautare mare
- Este un fel de best-first search ghidat de rezultatele unei simulari Monte-Carlo
- Metoda se bazeaza pe 2 ipoteze:
 - Adevarata **valoare a unei actiuni** (mutare in joc) poate fi aproximata utilizand simulari aleatoare
 - Valorile astfel obtinute pot fi utilizate pentru a **ajusta politica de selectie** spre o cea mai buna strategie
- MCTS (Coulom, 2006)

Monte Carlo Tree Search

- Baza metodei este o **unda de joc** (“**playout**”)
- **Playout** = un joc rapid jucat cu mutari aleatoare dintr-o anumita stare pana la sfarsitul jocului, obtinandu-se castig/pierdere sau un scor
- Fiecarui nod parcurs i se asociaza un merit
- In varainta cea mai simpla acest merit este un procent de castig = de cate ori s-a castigat daca s-a pornit unda din acel nod

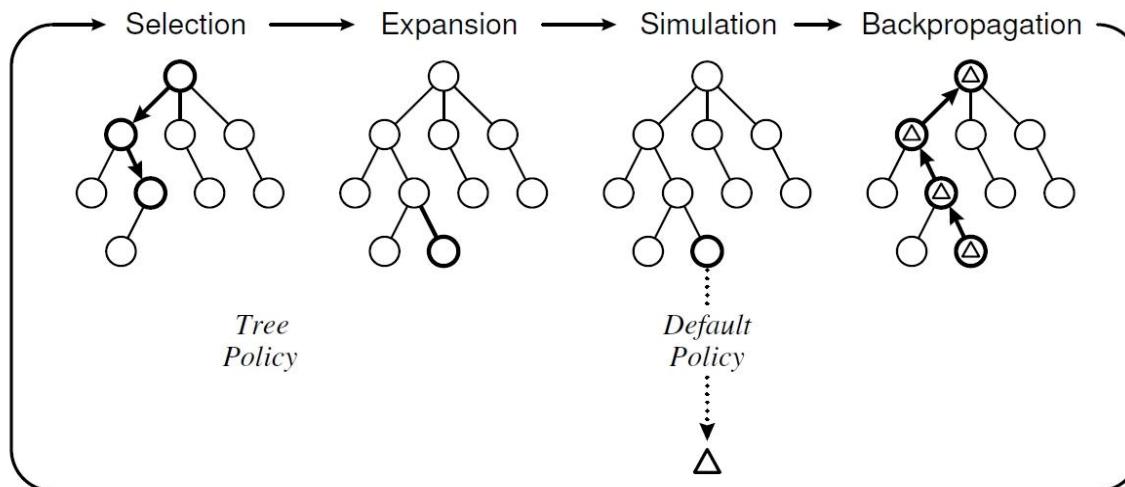
Monte Carlo Tree Search

- Algoritmul construieste progresiv un **arbore de joc parțial**, ghidat de rezultatele explorarilor anterioare ale acestui arbore
- Arborele este utilizat pentru a **estima valoarea miscarilor**, estimarile devenind din ce in ce mai bune pe masura ce arborele este construit
- Algoritmul implica construirea iterativa a arborelui de cautare pana cand o anumita cantitate de efort s-a atins, si intoarce cea mai buna actiune gasita

Monte Carlo Tree Search

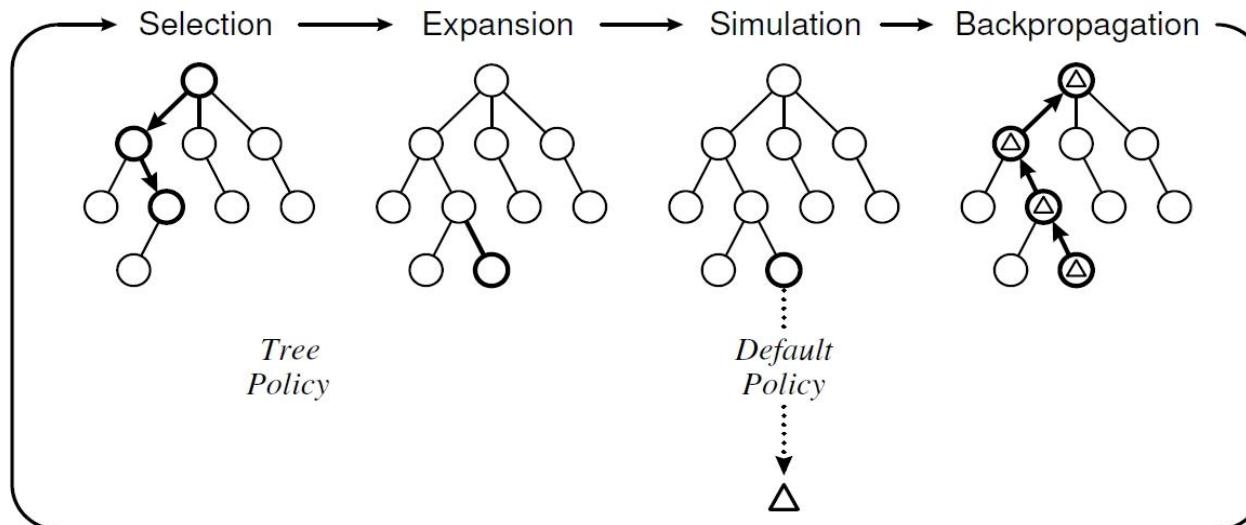
Pentru fiecare iteratie se aplica 4 pasi:

- **Selectie** – Pornind de la radacina o politica de selectie a copiilor este aplicata recursiv pana cand se gaseste cel mai interesant nod neexpandat (un nod E care are un copil ce nu este inca parte a arborelui)
- **Expandare** – Unul sau mai multe noduri copii a lui E sunt adaugate in arbore cf. actiunilor disponibile



Monte Carlo Tree Search

- **Simulare** – Se executa o simulare de la nodul/nodurile noi cf. politicii implicate pentru a obtine un rezultat R
- **Backpropagation** – rezultatul simulatiei este propagat inapoi catre nodurile care au fost parcuse si se actualizeaza valorile acestora



Monte Carlo Tree Search

Algoritm MCTS(Radacina) intoarce cea mai buna miscare

Creaza nodul radacina v_0 cu starea s_0

cat timp nu s-au epuizat resursele **repeta**

$v_1 \leftarrow \text{TreePolicy}(v_0)$

$\Delta \leftarrow \text{DefaultPolicy}(v_1)$

$\text{BackUp}(v_1, \Delta)$

intoarce $a(\text{BestChild}(v_0))$

- **TreePolicy** – construieste un nod frunza din nodurile aflate deja in arborele de cautare; nodul la care se ajunge cu TreePolicy este v_1
- **DefaultPolicy** – joaca jocul dintr-o stare neterminala v_1 pentru a produce o estimare a valorii / recompensa (pana stare terminala)
- **$a(\text{BestChild}(v_0))$** – actiunea care selecteaza cel mai bun copil a lui v_0

Algoritm MCTS(Radacina) intoarce cea mai buna miscare

Creaza nodul radacina v_0 cu starea s_0

cat timp nu s-au epuizat resursele **repeta**

$v_1 \leftarrow \text{TreePolicy}(v_0)$

$\Delta \leftarrow \text{DefaultPolicy}(v_1)$

$\text{BackUp}(v_1, \Delta)$

intoarce $a(\text{BestChild}(v_0))$

TreePolicy(v) intoarce un nod

cat timp v este nod neterminat **executa**

daca v nu este complet expandat **atunci** $v \leftarrow \text{Expand}(v)$; **break**

altfel $v \leftarrow \text{BestChild}(v)$

intoarce v

Expand(v) intoarce un nod

alege $a \in \text{actiunile neincercate inca din } A(v)$ /* $A(v)$ act legale in v */

adauga un copil nou v' la v cu $v' = \text{next}(v, a)$

intoarce v'

Algoritm MCTS(Radacina) intoarce cea mai buna miscare

Creaza nodul radacina v_0 cu starea s_0

cat timp nu s-au epuizat resursele **repeta**

$v_1 \leftarrow \text{TreePolicy}(v_0)$

$\Delta \leftarrow \text{DefaultPolicy}(v_1)$

$\text{BackUp}(v_1, \Delta)$

intoarce $a(\text{BestChild}(v_0))$

DefaultPolicy(s) intoarce recompensa

cat timp s este nod neterminal **executa**

alege aleator $a \in A(s)$

$s \leftarrow \text{next}(s, a)$

intoarce recompensa pentru starea s

BackUp(v, Δ)

cat timp v nu este null **executa**

$N(v) \leftarrow N(v) + 1$

/ numar vizitari nod v */*

$Q(v) \leftarrow Q(v) + \Delta(v)$

/ recompensa nod */*

$v \leftarrow p$

/ p parintele lui v */*

Monte Carlo Tree Search

Ce strategii se folosesc pt fiecare pas?

- **Selectia**
- Problema exploatarii vs explorare
 - Selectam stari din care s-a castigat des si au fost parcuse de multe ori
 - Ori selectam stari cu putine simulari anterioare
- Diferite strategii propuse in literatura

Monte Carlo Tree Search

Exemplu de strategie pentru selectie

Fie I multimea de noduri succesoare nodului curent p . Se selecteaza copilul K a nodului p care satisface formula

$$K \in \arg \max_{i \in I} \left(\frac{Q(i)}{N(i)} + C * \sqrt{\frac{2 * \ln N(p)}{N(i)}} \right)$$

pensa nodului i

Q(i) - recompensa nodului i

$N(i)$ – numarul de vizitari a nodului i

$N(p)$ – numarul de vizitari a nodului p (parinte)

C – coefficient experimental

BestChild(v) intoarece K

Monte Carlo Tree Search

Expandarea

Se genereaza un nou de unde se va incepe simularea

Simularea

- Total aleator, sau pseudoaleator, sau combinat cu euristici

Backpropagation

- Diferite metode

$$V_p = \frac{V_{med} * W_{med} + V_r * N_r}{W_{med} * N_r}$$

V_{med} – media valorilor nodurilor copii

W_{med} – ponderea acestei medii

V_r – mutarea cu cel mai mare numar de simulari

N_r – numarul de ori de care s-a jucat V_r

Monte Carlo Tree Search

- Converge spre valorile Minimax, dar lent
- Cu toate acestea mai eficient decat AlfaBeta
- Este un algoritm de tip anytime
- Din cauza naturii sale probabilistice, algoritmul nu gaseste întotdeauna cea mai buna mutare, dar are, în general, un succes rezonabil în cazul alegerii mutarilor care duc la sanse mari de câștig
- Poate să nu detecteze o ramură care conduce la pierdere (din cauza componentei aleatoare)
- Light playouts – aleator
- Heavy playouts – euristici pentru selectarea miscarii următoare (nu strict aleator)

Monte Carlo Tree Search

MoGo

A participat in 30 de turnee intre 2006 si 2010 (9 x 9 GO)

A castigat contra profesionistilor

- MoGo invinge pe campionul Myungwan Kim, august 2008, utilizand MCTS
- **FUEGO** - 2009 – a invins mai multi campioni la 9 x 9 GO



Monte Carlo Tree Search

2009 – MoHex devine campion la jocul Hex

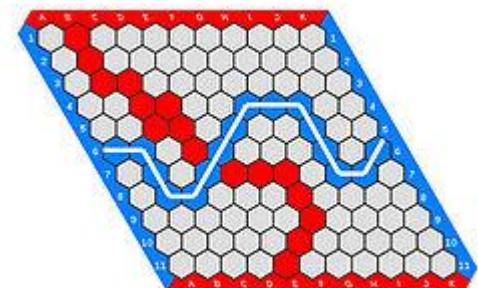
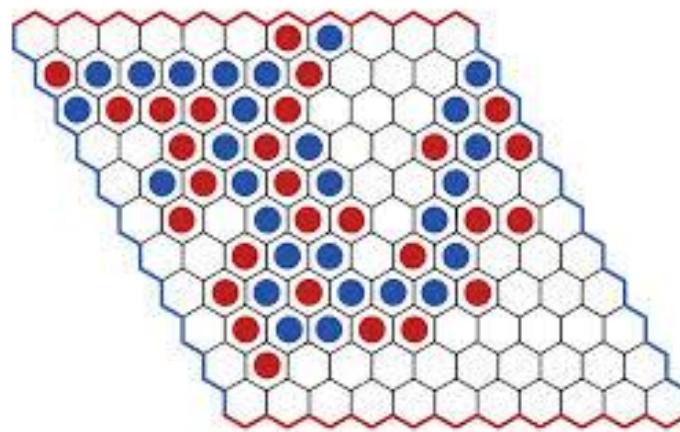
11 x 11

13 x 13

14 x 14

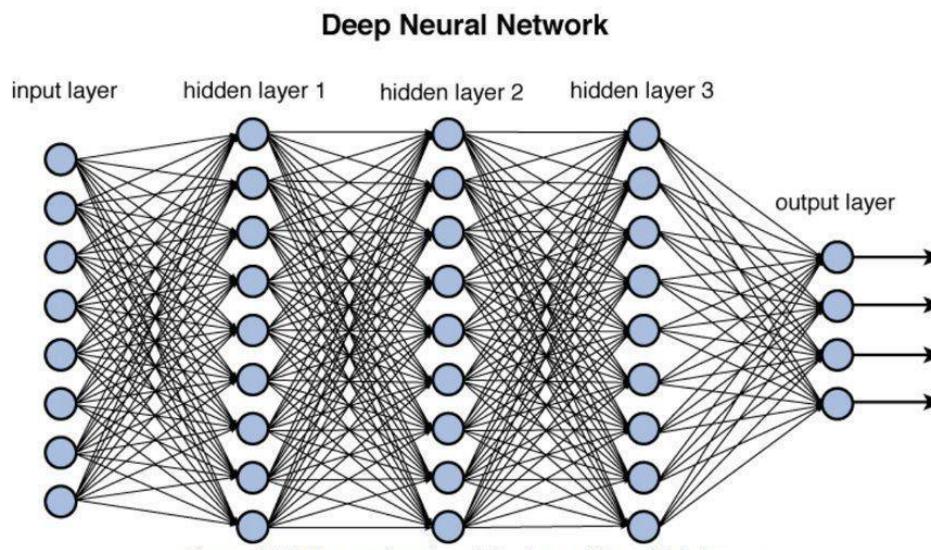
19 x 19

Joc inventat in 1940
de Piet Hein and John Nash



1.4 Deep Q-Learning pt jocuri

- Combina invatarea prin recompensa (Q-Learning) cu Deep Neural Networks (retele neurale adanci)



Q-Learning

Invatare prin recompensa

- **Functie actiune-valoare** = atribuie o utilitate estimata executarii unei actiuni intr-o stare
- $Q: A \times S \rightarrow U$
- Foloseste o matrice (stare,actiune) - valoare

Q-Learning

Foloseste **ecuatia lui Bellman**

$Q(a,s) \leftarrow$

$$Q(a,s) + \underline{\alpha}(R(s) + \delta \max_{a'} Q(a', s') - Q(a,s))$$

calculata dupa fiecare tranzitie din s in s' .

α - viteza de invatare

Q-Learning

1. Initializeaza δ si matricea \mathbf{R}
2. Initializeaza \mathbf{Q} la 0 sau aleator
3. Pentru fiecare episod repeta
 - Selecteaza starea initiala s aleator
 - **cat timp s nu este stare scop repeta**
 - obtine recompensa $\mathbf{R}(s)$
 - selecteaza o actiune a din starea s
 - executa trecerea in s' cu a
 - $\mathbf{Q}(a,s) \leftarrow \mathbf{Q}(a,s) + \alpha(\mathbf{R}(s) + \delta \max_a \mathbf{Q}(a', s') - \mathbf{Q}(a,s))$
 - $s \leftarrow s'$

Q-Learning

Utilizare Q

1. $s \leftarrow$ starea initiala
2. gaseste actiunea **a** care maximizeaza $Q(a,s)$
3. Executa **a**
4. **Repeta** de la 2 pana stare scop

Deep Q-Learning

- Combina Q-Learning cu DNN
- Reteaua neurala invata estimarea perechilor stare-actiune numai pe baza descrierii starii si a recompensei, fara a sti regulile de joc

Atari Deep Q-Learning

- Deep learning – extrage caracteristici de nivel inalt din imagini sau speech (vorbire)

Atari

- O retea convolutionala este antrenata sa invete evaluarea stare-actiune pt a aplica RL
- Intrare: pixelii din imagine
- Iesire: valori Q



Atari Deep Q-Learning

- Selecteaza actiunea
- Actiunea trimisa emulatorului de joc care modifica starea si scorul jocului
- Agentul primeste o imagine (vectori de pixeli) si recompensa = schimbarea scorului
- Considera o secventa de stari si observatii

$$S_t = s_1, a_1, s_2, a_2, \dots, a_{t-1}, s_t$$

si invata strategia in functie de astfel de sechente

Atari Deep Q-Learning

- Foloseste un Q-network cu parametrii θ care minimizeaza functia de eroare (**Loss**)

$$L(\theta) = \text{Estimare} [y - Q(s, a, \theta)]^2$$

$$y = R(s) + \delta \max_{a'} Q(s', a')$$

AlphaGo

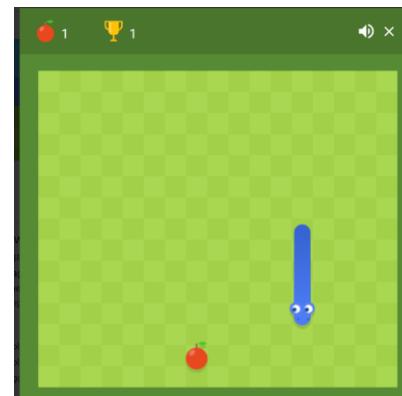
- Octombrie 2015 - AlphaGo joaca impotriva campionului european Fan Hui si castiga cu un scor de 5-0
- In 2016 castiga in fata lui Lee Sedol, detinator a 18 titluri mondiale
- A obtinut cel mai mare scor
- A inventat mutari
- Foloseste Deep Q-Learning

AlphaGo

- Foloseste:
 - MCTS
 - 2 retele neurale adanci: una pentru a invata politica de selectie si una pentru a invata evaluarea nodurilor
 - Algoritmul de Q-Learning – initializat cu politica invatata si imbunatatita in functie de recompensele viitoare

Un exemplu simplu

- Jocul Snake
- Se dau sistemului parametrii legati de stare si recompensa
- Nu are reguli de joc
- Trebuie sa maximizeze recompensa



Un exemplu simplu

- Tabela Q

| Stare | Right | Left | Up | Down |
|-------|-------|-------|------|------|
| 1 | 0 | 0.31 | 0.12 | 0.87 |
| 2 | 0.98 | -0,21 | 0.01 | 0.14 |

Se actualizeaza cu Q-Learning si ecuatia lui Bellman

$$Q(a,s) \leftarrow$$

$$Q(a,s) + \alpha(R(s) + \delta \max_a Q(a', s') - Q(a,s))$$

1. Initializeaza \mathbf{Q} aleator
 2. Initializeaza δ matricea \mathbf{R}
 3. Pentru fiecare episod repeta
 - Obtine starea curenta \mathbf{s}
 - Executa o actiune \mathbf{a} (aleator sau selectata de RNA)
 - Obtine recompensa $\mathbf{R}(\mathbf{s})$
 - Executa trecerea in \mathbf{s}' cu \mathbf{a}
- $\mathbf{Q}(\mathbf{a}, \mathbf{s}) \leftarrow$
- $$\mathbf{Q}(\mathbf{a}, \mathbf{s}) + \alpha (\mathbf{R}(\mathbf{s}) + \delta \max_{\mathbf{a}'} \mathbf{Q}(\mathbf{a}', \mathbf{s}') - \mathbf{Q}(\mathbf{a}, \mathbf{s}))$$
- $$- \mathbf{s} \leftarrow \mathbf{s}'$$

Reprezentare stare pentru retea

11 variabile boolene

- Daca este pericol in jurul sarpelui (Right, Left, Straight)
- Daca sarpele se misca Up, Down, Left, Right
- Daca mancarea este Up, Down, Left, Right
- RNA estimeaza actiunea cea mai buna pentru o anumita stare incercand sa maximizeze recompensa pe baza functiei de eroare (**Loss**)

$$L_i(\theta_i) = [R(s) + \delta \max_{a'} Q(s', a') - Q(s, a, \theta_i)]^2$$

Inteligentă Artificială

Universitatea Politehnica Bucuresti
Anul universitar 2020-2021

Adina Magda Florea



Curs nr. 5

Reprezentarea cunostintelor in IA **Modelul logicii simbolice**

- Reprezentarea cunostintelor in logica simbolica
- Logica propozitiilor
- Logica cu predicate de ordinul I
- Demonstrarea teoremelor prin respingere rezolutiva
- Legatura cu limbajul Prolog

1. Reprezentarea cunostintelor in LS

- Logica – avantaje
- Puterea de reprezentare a diverselor logici simbolice
- Conceptualizare + exprimarea in limbaj
- Limbaj formal: sintaxa, semantica
- Reguli de inferenta

2. Logica propozitiilor

2.1 Sintaxa

- Alfabet
- O formula bine formata in calculul propositional se defineste recursiv astfel:
 - (1) Un atom este o formula bine formata
 - (2) Daca P este formula bine formata, atunci $\sim P$ este formula bine formata.
 - (3) Daca P si Q sint formule bine formate atunci $P \wedge Q$, $P \vee Q$, $P \rightarrow Q$ si $P \leftrightarrow Q$ sint formule bine formate.
 - (4) Multimea formulelor bine formate este generata prin aplicarea repetata a regulilor (1)..(3) de un numar finit de ori.

2.2 Semantica

- Interpretare
- *Functia de evaluare a unei formule*
- Proprietatile fbf
 - Valida/tautologie
 - Realizabila
 - Inconsistenta
 - Formule echivalente

Semantica - cont

- O formula F este o *consecinta logica a unei formule P* daca F are valoarea adevarat in toate interpretarile in care P are valoarea adevarat.
- O formula F este *consecinta logica a unei multimi de formule* P_1, \dots, P_n daca formula F este adevarata in toate interpretarile in care P_1, \dots, P_n sunt adevarate.
- Consecinta logica se noteaza $P_1, \dots, P_n \Rightarrow F$.
- **Teorema.** Formula F este consecinta logica a unei multimi de formule P_1, \dots, P_n daca formula $P_1 \wedge \dots \wedge P_n \rightarrow F$ este valida.
- **Teorema.** Formula F este consecinta logica a unei multimi de formule P_1, \dots, P_n daca formula $P_1 \wedge \dots \wedge P_n \wedge \neg F$ este inconsistenta.

Legi de echivalenta

| | | | |
|------------------------------|---|---|--|
| Idempotenta | $P \vee P \equiv P$ | $P \wedge P \equiv P$ | |
| Asociativitate | $(P \vee Q) \vee R \equiv P \vee (Q \vee R)$ | $(P \wedge Q) \wedge R \equiv P \wedge (Q \wedge R)$ | |
| Comutativitate | $P \vee Q \equiv Q \vee P$ | $P \wedge Q \equiv Q \wedge P$ | $P \leftrightarrow Q \equiv Q \leftrightarrow P$ |
| Distributivitate | $P \wedge (Q \vee R) \equiv (P \wedge Q) \vee (P \wedge R)$ | $P \vee (Q \wedge R) \equiv (P \vee Q) \wedge (P \vee R)$ | |
| De Morgan | $\sim (P \vee Q) \equiv \sim P \wedge \sim Q$ | $\sim (P \wedge Q) \equiv \sim P \vee \sim Q$ | |
| Eliminarea implicatiei | $P \rightarrow Q \equiv \sim P \vee Q$ | | |
| Eliminarea implicatiei duble | $P \leftrightarrow Q \equiv (P \rightarrow Q) \wedge (Q \rightarrow P)$ | | |

2.3 Obtinerea de noi cunostinte

- Conceptualizare
- Reprezentare in limbaj
- Teoria modelului

$$KB \models_S x$$

- Teoria demonstratiei

$$KB \vdash_{\mathcal{R}} x$$

- Logici monotone
- Logici nemonotone

3.4 Reguli de inferenta

- *Modus Ponens*
$$\frac{\begin{matrix} P \\ P \rightarrow Q \end{matrix}}{Q}$$
- *Substitutia*
- *Regula înlantuirii*
$$\frac{\begin{matrix} P \rightarrow Q \\ Q \rightarrow R \end{matrix}}{P \rightarrow R}$$
- *Regula introducerii conjunctiei*
$$\frac{\begin{matrix} P \\ Q \end{matrix}}{P \wedge Q}$$
- *Regula transpozitiei*
$$\frac{P \rightarrow Q}{\sim Q \rightarrow \sim P}$$

Exemplu

- *Mihai are bani*
- *Masina este alba*
- *Masina este frumoasa*
- *Daca masina este alba sau masina este frumoasa si Mihai are bani atunci Mihai pleaca in vacanta*
- B
- A
- F
- $(A \vee F) \wedge B \rightarrow C$

3. Logica cu predicate de ordinul I

3.1 Sintaxa

Fie D un domeniu de valori. Un *termen* se defineste astfel:

- (1) O constanta este un termen cu valoare fixa apartinand domeniului D .
- (2) O variabila este un termen ce poate primi valori diferite din domeniul D .
- (3) Daca f este o functie de n argumente si $t_1,..t_n$ sunt termeni, atunci $f(t_1,..t_n)$ este termen.
- (4) Toti termenii sunt generati prin aplicarea regulilor (1)...(3).

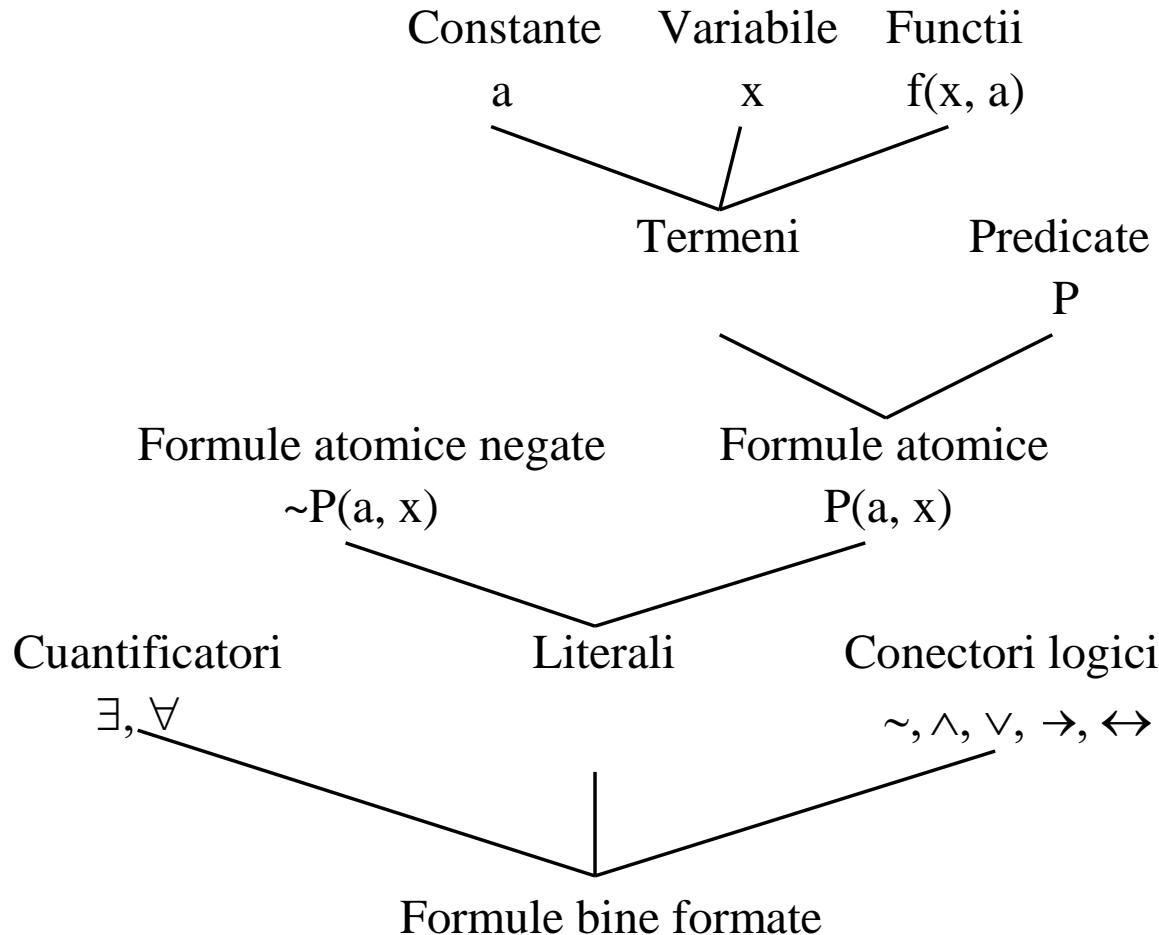
Sintaxa LP - cont

- Predicat de aritate n
- Atom sau formula atomica.
- Literal

O *formula bine formata* in logica cu predicate de ordinul I se defineste astfel:

- (1) Un atom este o formula bine formata
- (2) Daca $P[x]$ este fbf, atunci $\sim P[x]$ este fbf.
- (3) Daca $P[x]$ si $Q[x]$ sunt fbf atunci $P[x] \wedge Q[x]$,
 $P[x] \vee Q[x]$, $P \rightarrow Q$ si $P \leftrightarrow Q$ sunt fbf.
- (4) Daca $P[x]$ este fbf atunci $\forall x P[x]$, $\exists x P[x]$ sunt fbf.
- (5) Multimea formulelor bine formate este generata prin aplicarea repetata a regulilor (1)..(4) de un numar finit de ori.

Sintaxa pe scurt



FNC, FND

- O formula bine formata este in *forma normala conjunctiva*, pe scurt FNC, daca formula are forma

$$F_1 \wedge \dots \wedge F_n,$$

unde este F_i , $i=1,n$ sunt formule formate dintr-o disjunctie de literali ($L_{i1} \vee \dots \vee L_{im}$).

- O formula bine formata este in *forma normala disjunctiva*, pe scurt FND, daca formula are forma ,

$$F_1 \vee \dots \vee F_n,$$

unde F_i , $i=1,n$ sunt formule formate dintr-o conjunctie de literali ($L_{i1} \wedge \dots \wedge L_{im}$)

3.2 Semantica LP

- *Interpretarea unei formule F în logica cu predicate de ordinul I constă în fixarea unui domeniu de valori nevid D și a unei asignări de valori pentru fiecare constantă, funcție și predicat ce apar în F astfel:*
 - (1) Fiecarei constantă i se asociază un element din D.
 - (2) Fiecarei funcții f, de aritate n, i se asociază o corespondență $D^n \rightarrow D$, unde

$$D^n = \{(x_1, \dots, x_n) | x_1 \in D, \dots, x_n \in D\}$$

- (3) Fiecarui predicat de aritate n, i se asociază o corespondență $P: D^n \rightarrow \{a, f\}$

Interpretare I

$$(\forall x (((A(a,x) \vee B(f(x))) \wedge C(x)) \rightarrow D(x))$$

$$D = \{1, 2\}$$

| a | f(1) | f(2) | A(2,1) | A(2,2) | B(1) | B(2) | C(1) | C(2) | D(1) | D(2) |
|---|------|------|----------|----------|----------|----------|----------|----------|----------|----------|
| 2 | 2 | 1 | a | f | a | f | a | f | f | a |

$$X=1 \quad ((a \vee f) \wedge a) \rightarrow f$$

$$X=2 \quad ((f \vee a) \wedge f) \rightarrow a$$

3.3 Proprietatile fbf in LP

- Valida/tautologie
- Realizabila
- Inconsistenta
- Echivalente
- F - *consecinta logica a unei formule P*
- F - *consecinta logica a unei multimi de formule P₁,...P_n*
- **Teorema.** Formula F este consecinta logica a unei multimi de formule P₁,...P_n daca formula $P_1 \wedge \dots \wedge P_n \rightarrow F$ este valida.
- **Teorema.** Formula F este consecinta logica a unei multimi de formule P₁,...P_n daca formula $P_1 \wedge \dots \wedge P_n \wedge \neg F$ este inconsistenta.

Echivalenta cuantificatorilor

| | |
|---|---|
| $(Qx)F[x] \vee G \equiv (Qx)(F[x] \vee G)$ | $(Qx)F[x] \wedge G \equiv (Qx)(F[x] \wedge G)$ |
| $\sim ((\forall x)F[x]) \equiv (\exists x)(\sim F[x])$ | $\sim ((\exists x)F[x]) \equiv (\forall x)(\sim F[x])$ |
| $(\forall x)F[x] \wedge (\forall x)H[x] \equiv (\forall x)(F[x] \wedge H[x])$ | $(\exists x)F[x] \vee (\exists x)H[x] \equiv (\exists x)(F[x] \vee H[x])$ |
| $(Q_1x)F[x] \wedge (Q_2x)H[x] \equiv (Q_1x)(Q_2z)(F[x] \wedge H[z])$ | $(Q_1x)F[x] \vee (Q_2x)H[x] \equiv (Q_1x)(Q_2z)(F[x] \vee H[z])$ |

Exemple

- Exista un mar rosu
- Toate merele sunt rosii
- Toate obiectele sunt mere rosii

Toate ciupercile purpurii sunt otravitoare

- $\forall x (\text{Purpuriu}(x) \wedge \text{Ciuperca}(x)) \Rightarrow \text{Otravitor}(x)$
- $\forall x \text{ Purpuriu}(x) \Rightarrow (\text{Ciuperca}(x) \Rightarrow \text{Otravitor}(x))$
- $\forall x \text{ Ciuperca }(x) \Rightarrow (\text{Purpuriu }(x) \Rightarrow \text{Otravitor}(x))$

$(\forall x)(\exists y) \text{ iubeste}(x,y)$

$(\exists y)(\forall x) \text{ iubeste}(x,y)$

3.4. Reguli de inferenta in LP

- Modus Ponens (MP)

$$\frac{P(a) \quad (\forall x)(P(x) \rightarrow Q(x))}{Q(a)}$$

- Substitutia
- Regula inlantuirii
- Transpozitia
- Eliminarea conjunctiei (ElimC)
- Introducerea conjunctiei (IntrC)
- Instantierea universala (InstU)
- Instantierea existentiala (InstE)
- Rezolutia

Exemplu

- Caii sunt mai rapizi decat cainii si exista un ogar care este mai rapid decat orice iepure. Se stie ca Harry este un cal si ca Ralph este un iepure. Sa se demonstreze faptul ca Harry este mai rapid decat Ralph.
- $\text{Cal}(x)$ $\text{Ogar}(y)$
- $\text{Caine}(y)$ $\text{Iepure}(z)$
- $\text{MaiRapid}(y,z)$

$\forall x \forall y \text{Cal}(x) \wedge \text{Caine}(y) \rightarrow \text{MaiRapid}(x,y)$

$\exists y \text{Ogar}(y) \wedge (\forall z \text{Iepure}(z) \rightarrow \text{MaiRapid}(y,z))$

Cal(Harry)

Iepure(Ralph)

$\forall y \text{Ogar}(y) \rightarrow \text{Caine}(y)$

$\forall x \forall y \forall z \text{MaiRapid}(x,y) \wedge \text{MaiRapid}(y,z) \rightarrow \text{MaiRapid}(x,z)$

Exemplu de demonstrare

- **Teorema: MaiRapid(Harry, Ralph) ?**
- **Demonstrare folosind reguli de inferenta**

1. $\forall x \forall y \text{Cal}(x) \wedge \text{Caine}(y) \rightarrow \text{MaiRapid}(x,y)$
2. $\exists y \text{Ogar}(y) \wedge (\forall z \text{Iepure}(z) \rightarrow \text{MaiRapid}(y,z))$
3. $\forall y \text{Ogar}(y) \rightarrow \text{Caine}(y)$
4. $\forall x \forall y \forall z \text{MaiRapid}(x,y) \wedge \text{MaiRapid}(y,z) \rightarrow \text{MaiRapid}(x,z)$
5. $\text{Cal}(\text{Harry})$
6. $\text{Iepure}(\text{Ralph})$
7. $\text{Ogar}(\text{Greg}) \wedge (\forall z \text{Iepure}(z) \Rightarrow \text{MaiRapid}(\text{Greg},z))$ 2, InstE
8. $\text{Ogar}(\text{Greg})$ 7, ElimC
9. $\forall z \text{Iepure}(z) \Rightarrow \text{MaiRapid}(\text{Greg},z))$ 7, ElimC

Exemplu de demonstrare - cont

- 10. $\text{Iepure}(\text{Ralph}) \rightarrow \text{MaiRapid}(\text{Greg}, \text{Ralph})$ 9, InstU
- 11. $\text{MaiRapid}(\text{Greg}, \text{Ralph})$ 6,10, MP
- 12. $\text{Ogar}(\text{Greg}) \rightarrow \text{Caine}(\text{Greg})$ 3, InstU
- 13. $\text{Caine}(\text{Greg})$ 12, 8, MP
- 14. $\text{Cal}(\text{Harry}) \wedge \text{Caine}(\text{Greg}) \rightarrow \text{MaiRapid}(\text{Harry}, \text{Greg})$ 1, InstU
- 15. $\text{Cal}(\text{Harry}) \wedge \text{Caine}(\text{Greg})$ 5, 13, IntrC
- 16. $\text{MaiRapid}(\text{Harry}, \text{Greg})$ 14, 15, MP
- 17. $\text{MaiRapid}(\text{Harry}, \text{Greg}) \wedge \text{MaiRapid}(\text{Greg}, \text{Ralph}) \rightarrow \text{MaiRapid}(\text{Harry}, \text{Ralph})$ 4, InstU
- 18. $\text{MaiRapid}(\text{Harry}, \text{Greg}) \wedge \text{MaiRapid}(\text{Greg}, \text{Ralph})$ 16, 11, IntrC
- 19. $\text{MaiRapid}(\text{Harry}, \text{Ralph})$ 17, 18, MP

4. Demonstrarea teoremelor prin respingere rezolutiva

4.1 Forma standard (forma clauzala)

- Clauza
- Clauza Horn
- Clauza Horn definita
- Clauza vida
- Forme de notare
 - Forma clauzala $\sim A(x) \vee B(x)$
 - Forma clauzala multime $\{\sim A(x), B(x)\}$
 - Forma consecinta (Gentzer) $A(x) \Rightarrow B(x)$
 - Forma Prolog $B(x) :- A(x)$

Transformare in forma clauzala

Elimina implicatiile (simple sau duble)

Aduce negatiile in fata atomilor

Muta toti cuantificatorii in fata formulei (redenumire variabile daca este necesar)

Skolemizare

Transforma in FNC

Elimina cuantificatorii universali (se considera implicit in clauze)

Elimina conjunctii – set de clauze

Normalizeaza variabilele (daca este cazul)

Transformare in forma clauzala - exemplu

$$\forall x \forall y (\text{Horse}(x) \wedge \text{Dog}(y) \rightarrow \text{Faster}(x, y))$$

$$\forall x \forall y (\sim(\text{Horse}(x) \wedge \text{Dog}(y)) \vee \text{Faster}(x, y))$$

$$\forall x \forall y (\sim\text{Horse}(x) \vee \sim\text{Dog}(y) \vee \text{Faster}(x, y))$$

$$\sim\text{Horse}(x) \vee \sim\text{Dog}(y) \vee \text{Faster}(x, y)$$

$$R = \{\sim\text{Horse}(x) \vee \sim\text{Dog}(y) \vee \text{Faster}(x, y)\}$$

Transformare in forma clauzala - exemplu

$\exists y (\text{Greyhound}(y) \wedge (\forall z (\text{Rabbit}(z) \rightarrow \text{Faster}(y, z))))$

$\exists y (\text{Greyhound}(y) \wedge (\forall z (\sim \text{Rabbit}(z) \vee \text{Faster}(y, z))))$

$\exists y \forall z (\text{Greyhound}(y) \wedge (\sim \text{Rabbit}(z) \vee \text{Faster}(y, z)))$

$\forall z (\text{Greyhound}(\text{Greg}) \wedge (\sim \text{Rabbit}(z) \vee \text{Faster}(\text{Greg}, z)))$

$\text{Greyhound}(\text{Greg}) \wedge (\sim \text{Rabbit}(z) \vee \text{Faster}(\text{Greg}, z))$

$R = \{\text{Greyhound}(\text{Greg}), \sim \text{Rabbit}(z) \vee \text{Faster}(\text{Greg}, z)\}$

4.2 Unificarea expresiilor

- Substitutie α
- Unificator
- Cel mai general unificator β
- Expresie
- Algoritm de unificare

Algoritm Unifica(E1,E2): Unificarea expresiilor

1. **daca** E1 si E2 sunt constante

atunci

1.1. **daca** E1=E2

atunci intoarce { }

1.2. **intoarce INSUCCES**

2. **daca** E1 este variabila **atunci**

2.1 **daca** E1 apare in E2 **atunci intoarce INSUCCES**

altfel intoarce {E1/E2}

3. **daca** E2 este variabila **atunci**

3.1 **daca** E2 apare in E1 **atunci intoarce INSUCCES**

altfel intoarce {E2/E1}

4. **daca** E1= { } si E2 = { } **atunci intoarce SUCCES**

5. **daca** E1= { } sau E2 = { } **atunci intoarce INSUCCES**

6. dacă $E1 = \text{simb}(t_{11}, \dots, t_{1n})$ **și** $E2 = \text{simb}(t_{21}, \dots, t_{2n})$
atunci

6.1 $X \leftarrow t_{11}$, $Y \leftarrow t_{21}$

6.2 $\text{Rest}_1 \leftarrow t_{12}, \dots, t_{1n}$, $\text{Rest}_2 \leftarrow t_{22}, \dots, t_{2n}$

6.3 $\alpha_1 \leftarrow \text{Unifica}(X, Y)$

6.4 **dacă** $\alpha_1 = \text{INSUCCES}$

atunci **intoarce** INSUCCES

6.5 $G_1 \leftarrow \text{aplica}(\alpha_1, \text{Rest}_1)$, $G_2 \leftarrow \text{aplica}(\alpha_1, \text{Rest}_2)$

6.6 $\alpha_2 \leftarrow \text{Unifica}(G_1, G_2)$

6.7 **dacă** $\alpha_2 = \text{INSUCCES}$

atunci **intoarce** INSUCCES

altfel **intoarce** (α_1, α_2)

7. intoarce INSUCCES

sfarsit.

4.3 Rezolutia in logica propozitiilor

- Rezolvent
- Clauze care rezolva
- Principiul demonstratiei
- Arbore/graf de demonstrare

Algoritm: Respingerea prin rezolutie in logica propozitionala.

1. Converteste setul de axiome A in forma clauzala si obtine multimea de clauze S
2. Neaga teorema, transforma teorema negata in forma clauzala si adauga rezultatul la S
3. **repeta**
 - 3.1. Selecteaza o pereche de clauze C_1 si C_2 din S
 - 3.2. Determina $R = \text{Res}(C_1, C_2)$
 - 3.3. **daca** $R \neq \square$
atunci adauga R la S
pana $R = \square$ sau nu mai exista nici o pereche de clauze care rezolva
4. **daca** s-a obtinut clauza vida
atunci teorema este adevarata (este demonstrata)
5. **altfel** teorema este falsa
sfarsit.

- Comentarii
 - strategie
 - decidabilitate

4.4 Rezolutia in logica predicatelor

- Rezolvent binar
- Clauze care rezolva
- Factor al unei clauze
- Rezolvent

Algoritm: Respingerea prin rezolutie in logica predicatelor.

1. Converteste setul de axiome A in forma clauzala si obtine multimea de clauze S
 2. Neaga teorema, transforma teorema negata in forma clauzala si adauga rezultatul la S
 3. **repeta**
 - 3.1. Selecteaza o pereche de clauze C_1 si C_2 din S
 - 3.2. Determina $R = \{\text{Res}(C_1, C_2)\}$
 - 3.3. **daca** $\square \notin R$
atunci reuneste R cu S
pana s-a obtinut \square **sau** nu mai exista nici o pereche de clauze care rezolva **sau** un efort a fost epuizat
 4. **daca** s-a obtinut clauza vida
atunci teorema este adevarata (este demonstrata)
 5. **altfel daca** nu mai exista nici o pereche de clauze care rezolva
atunci teorema este falsa
altfel nu se stie
- sfarsit.**

- Comentarii
 - strategie
 - decidabilitate
 - completitudine

Exemplu

- Horses are faster than dogs and there is a greyhound that is faster than every rabbit. We know that Harry is a horse and that Ralph is a rabbit. Derive that Harry is faster than Ralph.

- Horse(x) Greyhound(y)
- Dog(y) Rabbit(z)
- Faster(y,z))

$\forall x \forall y \text{Horse}(x) \wedge \text{Dog}(y) \rightarrow \text{Faster}(x,y)$

$\exists y \text{Greyhound}(y) \wedge (\forall z \text{Rabbit}(z) \rightarrow \text{Faster}(y,z))$

Horse(Harry)

Rabbit(Ralph)

$\forall y \text{Greyhound}(y) \rightarrow \text{Dog}(y)$

$\forall x \forall y \forall z \text{Faster}(x,y) \wedge \text{Faster}(y,z) \rightarrow \text{Faster}(x,z)$

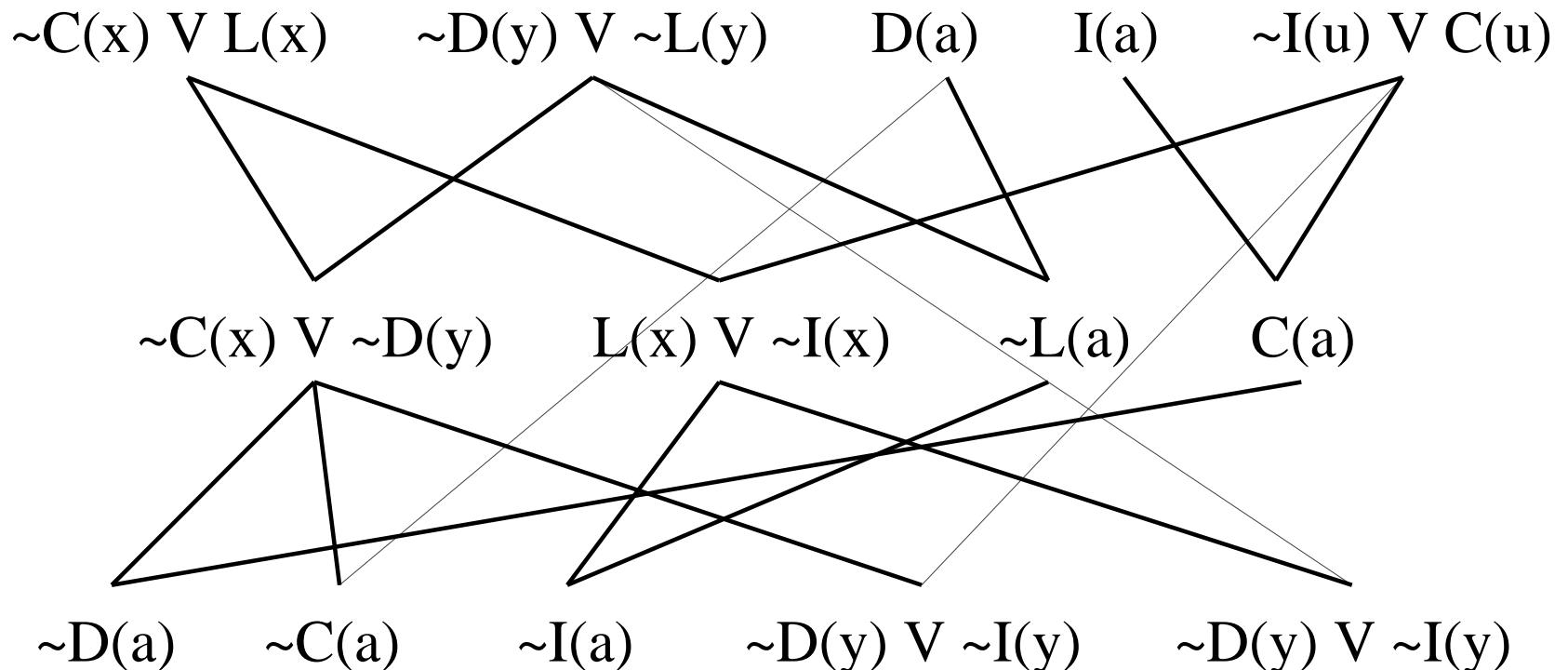
- A1.** $\forall x \forall y (\text{Horse}(x) \wedge \text{Dog}(y) \rightarrow \text{Faster}(x,y))$
- A2.** $\exists y \text{Greyhound}(y) \wedge (\forall z \text{Rabbit}(z) \rightarrow \text{Faster}(y,z))$
- A3.** $\text{Horse}(\text{Harry})$
- A4.** $\text{Rabbit}(\text{Ralph})$
- A5.** $\forall y (\text{Greyhound}(y) \rightarrow \text{Dog}(y))$
- A6.** $\forall x \forall y \forall z (\text{Faster}(x,y) \wedge \text{Faster}(y,z) \rightarrow \text{Faster}(x,z))$
- T** $\text{Faster}(\text{Harry},\text{Ralph})$

- C1.** $\neg \text{Horse}(x) \vee \neg \text{Dog}(y) \vee \text{Faster}(x,y)$
- C2.** $\text{Greyhound}(\text{Greg})$
- C2'** $\neg \text{Rabbit}(z) \vee \text{Faster}(\text{Greg},z)$
- C3.** $\text{Horse}(\text{Harry})$
- C4.** $\text{Rabbit}(\text{Ralph})$
- C5.** $\neg \text{Greyhound}(y) \vee \text{Dog}(y)$
- C6.** $\neg \text{Faster}(x,y) \vee \neg \text{Faster}(y,z) \vee \text{Faster}(x,z)$
- C7.** $\neg \text{Faster}(\text{Harry},\text{Ralph})$

4.5 Strategii rezolutive

- *Strategia dezvoltarii pe latime*
- *Strategia multimii suport*
- *Strategia rezolutiei liniare*
- *Strategia rezolutiei liniare de intrare*
- *Strategia rezolutiei unitare*
- *Strategia eliminarii*

Strategia dezvoltării pe latime



Strategia dezvoltării pe latime

- S_0, L_0
- $L_{k+1} \leftarrow \{ \text{Res}(C_i, C_j) | C_i \in L_k, C_j \in S_k \}$
- $S_{k+1} \leftarrow S_k \cup L_{k+1}$
 $k = 0, 1, 2, \dots$

Strategia multimii suport

- S, T $S_0 = S \cup T, \quad S \cap T = \emptyset$
- $S \leftarrow S \cup \{ \text{Res}(C_i, C_j) | C_i \in S, C_j \in T \}$

Strategia rezolutiei liniare

- $S, C_0 \in S$
- $C_1 \leftarrow \{Res(C_0, C_i) | C_0, C_i \in S\}$
- $C_{k+1} \leftarrow \{Res(C_k, C_i) | C_i \in \{C_{k-1}, C_{k-2}, \dots\} \cup S\}$
 $k=1, 2, 3, \dots$

Strategia rezolutiei liniare de intrare

- $S, C_0 \in S$
- $C_1 \leftarrow \{Res(C_0, C_i) | C_0, C_i \in S\}$
- $C_{k+1} \leftarrow \{Res(C_k, C_i) | C_i \in S\}$
 $k=1, 2, 3, \dots$

Strategia rezolutiei unitare

Clauze subsumate

- O clauza C subsumeaza o clauza D daca si numai daca exista o substitutie α astfel incat $C\alpha \subseteq D$. D se numeste clauza subsumata.
- $C=P(x)$ $D=P(a) \vee Q(a)$

Strategia eliminarii

Verificare C subsumeaza D

$$D = L_1 \vee \dots \vee L_m, \quad \alpha = \{x_1/a_1, \dots, x_n/a_n\}$$

$$D\alpha = L_1\alpha \vee \dots \vee L_m\alpha, \quad \neg D\alpha = \neg L_1\alpha \wedge \dots \wedge \neg L_m\alpha$$

1. $W \leftarrow \{\neg L_1\alpha, \dots, \neg L_m\alpha\}$
2. $k \leftarrow 0, U^0 \leftarrow \{C\}$
3. **daca** $\square \in U^k$ **atunci intoarce SUCCES**
4. $U^{k+1} \leftarrow \{\text{rez}(C_1, C_2) \mid C_1 \in U^k, C_2 \in W\}$
5. **daca** $U^{k+1} = \emptyset$ **atunci intoarce INSUCCES**
6. $k \leftarrow k+1$
7. **repeta de la 3**

sfarsit

5. Legatura cu limbajul Prolog

- R. Kowalski, A. Colmerauer - începutul anilor '70
- Corespondenta Logica cu Predicate si limbajul Prolog – *clauze Horn definite*
- Structura logica vs structura de control si executie

5.1 Structura logica a limbajului Prolog

- Un scop **S** este **adevarat** intr-un program Prolog, adica *poate fi satisfacut sau derivă logic din program*, daca si numai daca:
 1. exista o clauza **C** a programului;
 2. exista o instanta **I** a clauzei **C** astfel incat:
 - antetul lui **I** sa fie identic cu cel al lui **S**;
 - toate scopurile din corpul lui **I** sunt adevarate, deci pot fi satisfacute

5.2 Structura de control si executie a limbajului Prolog

- Rezolutie liniara de intrare
- Ordinea faptelor si clauzelor
- Ordinea scopurilor in corpul clauzelor
- Backtracking

5.3 Prolog – exemplu

```
% parinte(IndividX, IndividY)
```

```
% stramos(IndividX, IndividZ)
```

```
parinte(vali, gelu).
```

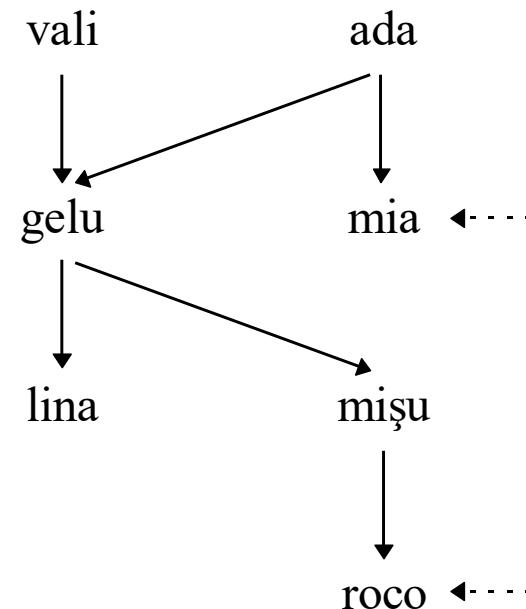
```
parinte(ada, gelu).
```

```
parinte(ada, mia).
```

```
parinte(gelu, lina).
```

```
parinte(gelu, misu).
```

```
parinte(misu, roco).
```



str1(X, Z) :- parinte(X, Z).

str1(X, Z) :- parinte(X, Y), str1(Y, Z).

Prolog – exemplu

% Se schimba ordinea regulilor:

str2(X, Z) :- parinte(X, Y), str2(Y, Z).
str2(X, Z) :- parinte(X, Z).

% Se schimba ordinea scopurilor in prima varianta:

str3(X, Z) :- parinte(X, Z).
str3(X, Z) :- str3(X, Y), parinte(Y, Z).

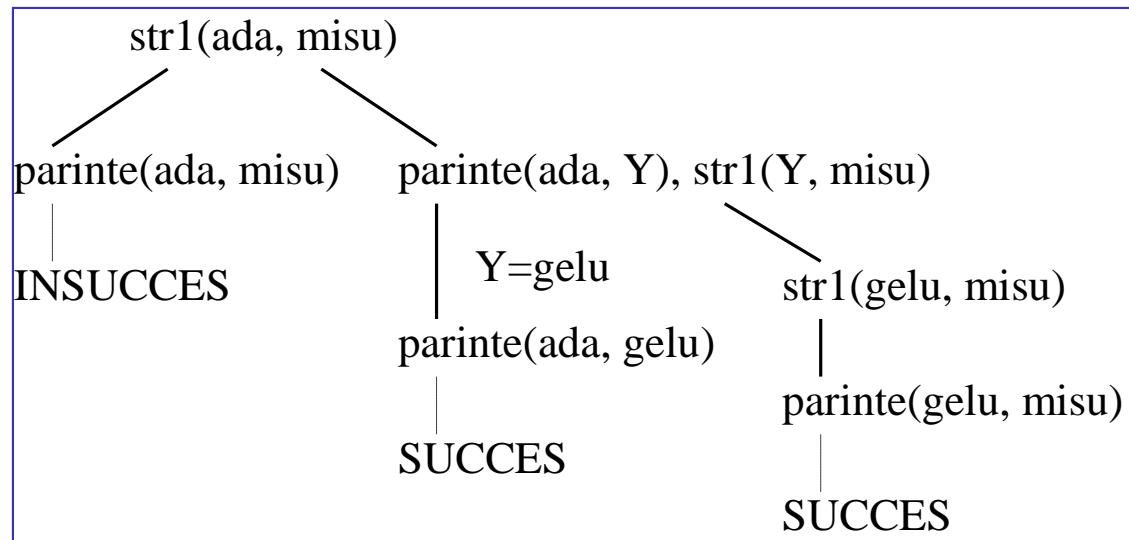
% Se schimba atat ordinea regulilor, cat si ordinea scopurilor:

str4(X, Z) :- str4(X, Y), parinte(Y, Z).
str4(X, Z) :- parinte(X,Z).

Prolog - exemplu

?- **str1(ada, misu).**

yes

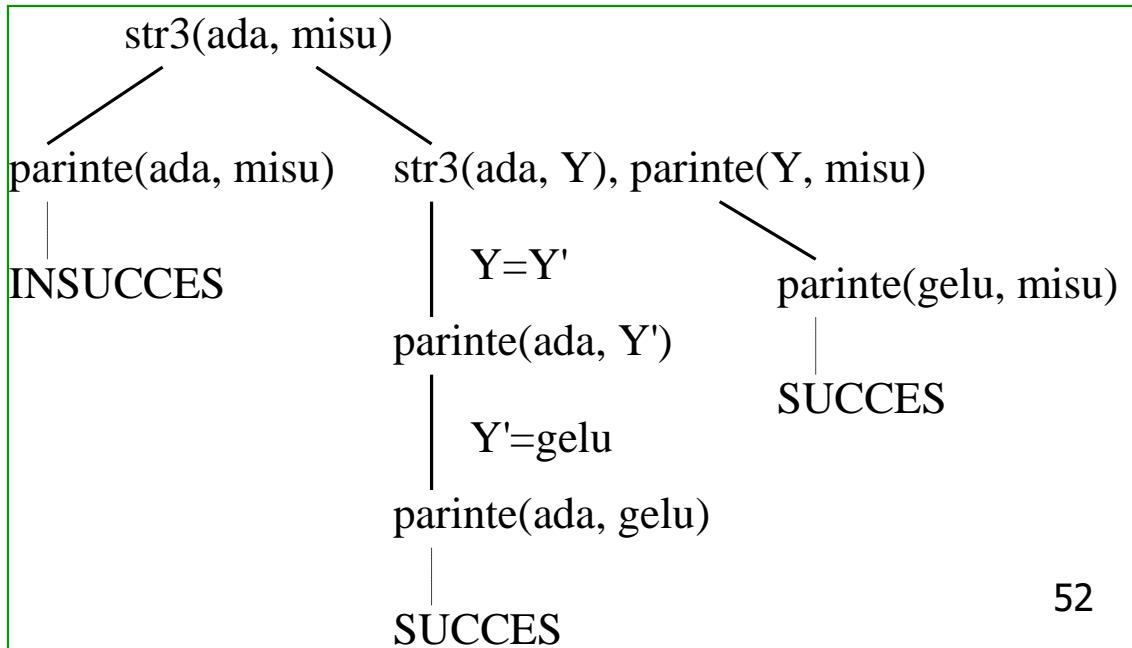


?- **str2(ada, misu).**

yes

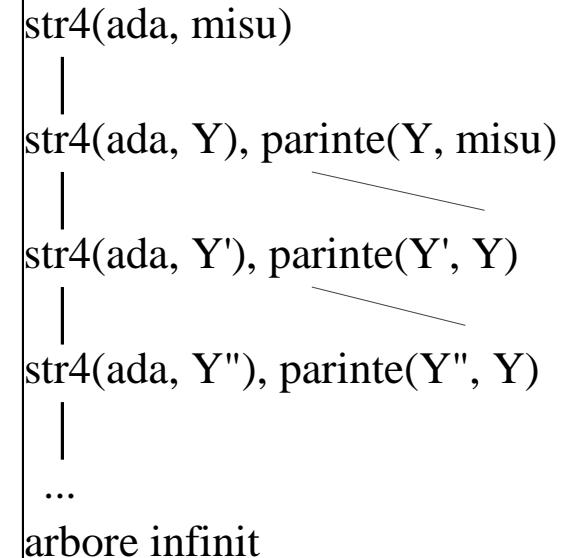
?- **str3(ada, misu).**

yes



Prolog – exemplu

?- str4(ada, misu).



?- str3(mia, roco).

