

Paradigme de Programare

Conf. dr. ing. Andrei Olaru

andrei.olaru@upb.ro | cs@andreiolaru.ro
Departamentul de Calculatoare

2022

Cursul 13: Mașina algoritmică Markov

transform from \rightarrow to
then \rightarrow .



- 1 Introducere
- 2 Mașina algoritmică Markov
- 3 Aplicații

Introducere



- Model de calculabilitate efectivă, **echivalent** cu Mașina Turing și Calculul Lambda;
- Principiul de funcționare: *pattern matching* + substituție;
- Fundamentul teoretic al paradigmei **asociative** și al limbajelor bazate pe **reguli** (de forma *dacă-atunci*).



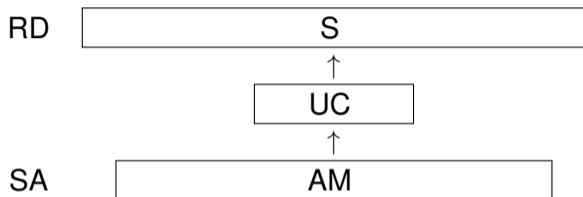
- Potrivită mai ales în cazul problemelor ce **nu** admit o soluție precisă algoritmică (ieftină);
- Codificarea **cunoștințelor** specifice unui domeniu și aplicarea lor într-o manieră **euristică**;
- Descrierea **proprietăților** soluției, prin contrast cu pașii care trebuie realizați pentru obținerea acesteia (**ce** trebuie obținut vs. **cum**);
- Absența unui flux explicit de control, deciziile fiind determinate, implicit, de cunoștințele valabile la un anumit moment → **data-driven control**.

Mașina algoritmică Markov



(implementări fără variabile generice)

- **Windows / Wine:** [<http://yad-studio.github.io/>]
- **mai multe:** [http://en.wikipedia.org/wiki/Markov_algorithm#External_links]



- Registrul de **date**, RD, cu secvența de **simboluri**, S
 - RD nemărginit la dreapta
 - $S \in (A_b \cup A_l)^*$, $A_b \cap A_l = \emptyset$ – alfabet de bază și de lucru
- Unitatea de **control**, UC
- Spațiul de stocare a **algoritmului**, SA, ce conține algoritmul Markov, AM
 - format din **reguli**.



- Unitatea de bază a unui algoritm Markov → **regula** asociativă de substituție:

şablon **identificare** (LHS) → şablon **substituție** (RHS)

- Exemplu: $ag_1c \rightarrow ac$
- **şabloanele** → secvențe de simboluri:
 - **constante**: simboluri din A_b
 - variabile **locale**: simboluri din A_l
 - variabile **generice**: simboluri speciale, din mulțimea G , legați la simboluri din A_b
- Dacă RHS este “.” → regulă **terminală**, ce încheie execuția maşinii (halt).



- De obicei, notate cu g , urmat de un indice;
- Mulțimea valorilor pe care le poate lua o variabilă → **domeniul** variabilei –
 $\text{Dom}(g) \subseteq A_b \cup A_l$;
- Legate la exact **un simbol** la un moment dat;
- **Durata de viață (scope)** → timpul aplicării regulii – sunt legate la identificarea șablonului și legarea se pierde după înlocuirea șablonului de identificare cu cel de substituție;
- Utilizabile în RHS **doar** în cazul apariției în LHS.

Structura Mașinii Markov

Algoritm Markov



- Mulțime **ordonată** de **reguli**, îmbogățite cu **declarații**:
 - de partiționare a mulțimii A_b
 - de variabile generice

Ex | **Exemplu** Eliminarea din dintr-un șir de simboluri din mulțimea $A \cup B$ simbolurilor ce aparțin mulțimii B :



- Mulțime **ordonată** de **reguli**, îmbogățite cu **declarații**:
 - de partiționare a mulțimii A_b
 - de variabile generice

Ex Exemplu Eliminarea din dintr-un șir de simboluri din mulțimea $A \cup B$ simbolurilor ce aparțin mulțimii B :

```
1 setDiff1(A, B); A g1; B g2;
2     ag2 -> a;
3     ag1 -> g1a;
4     a -> .;
5     -> a;
6 end
```

```
1 setDiff2(A, B); B g2;
2     g2 -> ;
3     -> .;
4 end
```

- $A, B \subseteq A_b$
- $g_1, g_2 \rightarrow$ variabile generice
- a nedeclarată \rightarrow variabilă locală ($a \in A_l$)



+ **Aplicabilitatea unei reguli** Regula $r : a_1 \dots a_n \rightarrow b_1 \dots b_m$ este aplicabilă dacă și numai dacă există un **subșir** $c_1 \dots c_n$, în RD, astfel încât $\forall i = \overline{1, n}$ **exact 1** condiție din cele de mai jos este îndeplinită:

- $a_i \in A_b \cup A_1 \wedge a_i = c_i$
- $a_i \in G \wedge c_i \in \text{Dom}(a_i) \wedge (\forall j = \overline{1, n} . a_j = a_i \Rightarrow c_j = c_i),$
- oriunde mai apare aceeași variabilă generică în șablonul de identificare, în poziția corespunzătoare din subșir avem același simbol.



+ Aplicarea regulii

$r : a_1 \dots a_n \rightarrow b_1 \dots b_m$ asupra unui subșir

$s : c_1 \dots c_n$, în raport cu care este **aplicabilă**, constă în **substituirea** lui s prin subșirul $q_1 \dots q_m$, calculat astfel încât pentru $\forall i = \overline{1, n}$:

- $b_i \in A_b \cup A_1 \Rightarrow q_i = b_i$
- $b_i \in G \wedge (\exists j = \overline{1, n} . b_i = a_j) \Rightarrow q_i = c_j$



Ex Exemplu

- $A_b = \{1, 2, 3\}$
- $A_1 = \{x, y\}$
- $\text{Dom}(g_1) = \{2\}$
- $\text{Dom}(g_2) = A_b$
- $S = 1111112x2y31111$
- $r : 1g_1xg_1yg_2 \rightarrow 1g_2x$



Exemplu

- $A_b = \{1, 2, 3\}$
- $A_1 = \{x, y\}$
- $\text{Dom}(g_1) = \{2\}$
- $\text{Dom}(g_2) = A_b$
- $S = 1111112x2y31111$

• $r : 1g_1xg_1yg_2 \rightarrow 1g_2x$

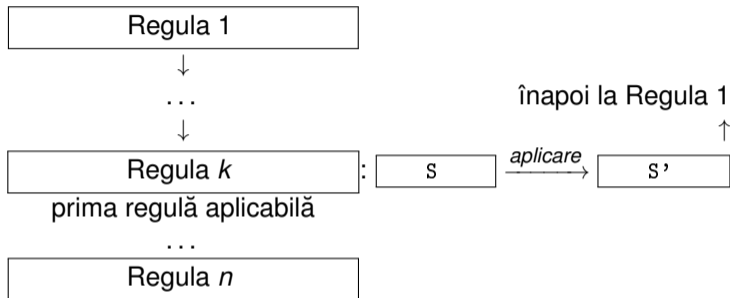
$S = 11111 \quad 1 \quad 2 \quad x \quad 2 \quad y \quad 3 \quad 1111$

$r : \quad \quad \quad 1 \quad g_1 \quad x \quad g_1 \quad y \quad g_2 \quad \rightarrow \quad 1g_2x$

$S' = 1111113x1111$



- Cazuri speciale: aplicabilitatea:
 - unei reguli pentru mai multe subșiruri;
 - mai multor reguli pentru același subșir.
- La un anumit moment, putem aplica propriu-zis o singură regulă asupra unui singur subșir;
- Nedeterminism inerent, ce trebuie exploatat, sau rezolvat;
- Convenție care poate fi făcută:
 - aplicarea primei reguli aplicabile, asupra
 - celui mai din stânga subșir asupra căreia este aplicabilă





Ex Inversarea intrării

- Ideea: mutarea, **pe rând**, a fiecărui element în poziția corespunzătoare. Mutarea se face prin pași incrementali de interschimbare a elementelor învecinate.



Ex Inversarea intrării

- Ideea: mutarea, **pe rând**, a fiecărui element în poziția corespunzătoare. Mutarea se face prin pași incrementali de interschimbare a elementelor învecinate.

```
1 Reverse(A); A g1, g2;  
2     ag1g2 -> g2ag1;  
3     ag1 -> bg1;  
4     abg1 -> g1a;  
5     a -> .;  
6     -> a;  
7 end
```

- DOP



Ex Inversarea intrării

- Ideea: mutarea, **pe rând**, a fiecărui element în poziția corespunzătoare. Mutarea se face prin pași incrementali de interschimbare a elementelor învecinate.

```
1 Reverse(A); A g1, g2;  
2     ag1g2 -> g2ag1;  
3     ag1 -> bg1;  
4     abg1 -> g1a;  
5     a -> .;  
6     -> a;  
7 end
```

- DOP $\xrightarrow{6}$ aDOP



Ex Inversarea intrării

- Ideea: mutarea, **pe rând**, a fiecărui element în poziția corespunzătoare. Mutarea se face prin pași incrementali de interschimbare a elementelor învecinate.

```
1 Reverse(A); A g1, g2;  
2     ag1g2 -> g2ag1;  
3     ag1 -> bg1;  
4     abg1 -> g1a;  
5     a -> .;  
6     -> a;  
7 end
```

- $DOP \xrightarrow{6} aDOP \xrightarrow{2} 0aDP$



Ex Inversarea intrării

- Ideea: mutarea, **pe rând**, a fiecărui element în poziția corespunzătoare. Mutarea se face prin pași incrementali de interschimbare a elementelor învecinate.

```
1 Reverse(A); A g1, g2;  
2     ag1g2 -> g2ag1;  
3     ag1 -> bg1;  
4     abg1 -> g1a;  
5     a -> .;  
6     -> a;  
7 end
```

- $DOP \xrightarrow{6} aDOP \xrightarrow{2} 0aDP \xrightarrow{2} OPaD$



Ex Inversarea intrării

- Ideea: mutarea, **pe rând**, a fiecărui element în poziția corespunzătoare. Mutarea se face prin pași incrementali de interschimbare a elementelor învecinate.

```
1 Reverse(A); A g1, g2;  
2     ag1g2 -> g2ag1;  
3     ag1 -> bg1;  
4     abg1 -> g1a;  
5     a -> .;  
6     -> a;  
7 end
```

- $DOP \xrightarrow{6} aDOP \xrightarrow{2} 0aDP \xrightarrow{2} OPaD \xrightarrow{3} OPbD$



Ex Inversarea intrării

- Ideea: mutarea, **pe rând**, a fiecărui element în poziția corespunzătoare. Mutarea se face prin pași incrementali de interschimbare a elementelor învecinate.

```
1 Reverse(A); A g1, g2;  
2     ag1g2 -> g2ag1;  
3     ag1 -> bg1;  
4     abg1 -> g1a;  
5     a -> .;  
6     -> a;  
7 end
```

- $DOP \xrightarrow{6} aDOP \xrightarrow{2} 0aDP \xrightarrow{2} OPaD \xrightarrow{3} OPbD \xrightarrow{6} aOPbD$



Ex Inversarea intrării

- Ideea: mutarea, **pe rând**, a fiecărui element în poziția corespunzătoare. Mutarea se face prin pași incrementali de interschimbare a elementelor învecinate.

```
1 Reverse(A); A g1, g2;  
2     ag1g2 -> g2ag1;  
3     ag1 -> bg1;  
4     abg1 -> g1a;  
5     a -> .;  
6     -> a;  
7 end
```

- $DOP \xrightarrow{6} aDOP \xrightarrow{2} 0aDP \xrightarrow{2} OPaD \xrightarrow{3} OPbD \xrightarrow{6} aOPbD \xrightarrow{2} Pa0bD$



Ex Inversarea intrării

- Ideea: mutarea, **pe rând**, a fiecărui element în poziția corespunzătoare. Mutarea se face prin pași incrementali de interschimbare a elementelor învecinate.

```
1 Reverse(A); A g1, g2;  
2     ag1g2 -> g2ag1;  
3     ag1 -> bg1;  
4     abg1 -> g1a;  
5     a -> .;  
6     -> a;  
7 end
```

- $DOP \xrightarrow{6} aDOP \xrightarrow{2} 0aDP \xrightarrow{2} OPaD \xrightarrow{3} OPbD \xrightarrow{6} aOPbD \xrightarrow{2} Pa0bD \xrightarrow{3} Pb0bD$



Ex Inversarea intrării

- Ideea: mutarea, **pe rând**, a fiecărui element în poziția corespunzătoare. Mutarea se face prin pași incrementali de interschimbare a elementelor învecinate.

```
1 Reverse(A); A g1, g2;  
2     ag1g2 -> g2ag1;  
3     ag1 -> bg1;  
4     abg1 -> g1a;  
5     a -> .;  
6     -> a;  
7 end
```

- $DOP \xrightarrow{6} aDOP \xrightarrow{2} 0aDP \xrightarrow{2} OPaD \xrightarrow{3} OPbD \xrightarrow{6} aOPbD \xrightarrow{2} PaObD \xrightarrow{3} PbObD$
 $\xrightarrow{6} aPbObD$



Ex Inversarea intrării

- Ideea: mutarea, **pe rând**, a fiecărui element în poziția corespunzătoare. Mutarea se face prin pași incrementali de interschimbare a elementelor învecinate.

```
1 Reverse(A); A g1, g2;  
2     ag1g2 -> g2ag1;  
3     ag1 -> bg1;  
4     abg1 -> g1a;  
5     a -> .;  
6     -> a;  
7 end
```

- $DOP \xrightarrow{6} aDOP \xrightarrow{2} 0aDP \xrightarrow{2} OPaD \xrightarrow{3} OPbD \xrightarrow{6} aOPbD \xrightarrow{2} PaObD \xrightarrow{3} PbObD$
 $\xrightarrow{6} aPbObD \xrightarrow{3} bPbObD$



Ex Inversarea intrării

- Ideea: mutarea, **pe rând**, a fiecărui element în poziția corespunzătoare. Mutarea se face prin pași incrementali de interschimbare a elementelor învecinate.

```
1 Reverse(A); A g1, g2;  
2     ag1g2 -> g2ag1;  
3     ag1 -> bg1;  
4     abg1 -> g1a;  
5     a -> .;  
6     -> a;  
7 end
```

- $DOP \xrightarrow{6} aDOP \xrightarrow{2} 0aDP \xrightarrow{2} OPaD \xrightarrow{3} OPbD \xrightarrow{6} aOPbD \xrightarrow{2} PaObD \xrightarrow{3} PbObD$
 $\xrightarrow{6} aPbObD \xrightarrow{3} bPbObD \xrightarrow{6} abPbObD$



Ex Inversarea intrării

- Ideea: mutarea, **pe rând**, a fiecărui element în poziția corespunzătoare. Mutarea se face prin pași incrementali de interschimbare a elementelor învecinate.

```
1 Reverse(A); A g1, g2;  
2     ag1g2 -> g2ag1;  
3     ag1 -> bg1;  
4     abg1 -> g1a;  
5     a -> .;  
6     -> a;  
7 end
```

- $DOP \xrightarrow{6} aDOP \xrightarrow{2} 0aDP \xrightarrow{2} OPaD \xrightarrow{3} OPbD \xrightarrow{6} aOPbD \xrightarrow{2} PaObD \xrightarrow{3} PbObD$
 $\xrightarrow{6} aPbObD \xrightarrow{3} bPbObD \xrightarrow{6} abPbObD \xrightarrow{4} PabObD$



Ex Inversarea intrării

- Ideea: mutarea, **pe rând**, a fiecărui element în poziția corespunzătoare. Mutarea se face prin pași incrementali de interschimbare a elementelor învecinate.

```
1 Reverse(A); A g1, g2;  
2     ag1g2 -> g2ag1;  
3     ag1 -> bg1;  
4     abg1 -> g1a;  
5     a -> .;  
6     -> a;  
7 end
```

- $DOP \xrightarrow{6} aDOP \xrightarrow{2} 0aDP \xrightarrow{2} OPaD \xrightarrow{3} OPbD \xrightarrow{6} aOPbD \xrightarrow{2} PaObD \xrightarrow{3} PbObD$
 $\xrightarrow{6} aPbObD \xrightarrow{3} bPbObD \xrightarrow{6} abPbObD \xrightarrow{4} PabObD \xrightarrow{4} POabD$



Ex Inversarea intrării

- Ideea: mutarea, **pe rând**, a fiecărui element în poziția corespunzătoare. Mutarea se face prin pași incrementali de interschimbare a elementelor învecinate.

```
1 Reverse(A); A g1, g2;  
2     ag1g2 -> g2ag1;  
3     ag1 -> bg1;  
4     abg1 -> g1a;  
5     a -> .;  
6     -> a;  
7 end
```

- $DOP \xrightarrow{6} aDOP \xrightarrow{2} 0aDP \xrightarrow{2} OPaD \xrightarrow{3} OPbD \xrightarrow{6} aOPbD \xrightarrow{2} PaObD \xrightarrow{3} PbObD$
 $\xrightarrow{6} aPbObD \xrightarrow{3} bPbObD \xrightarrow{6} abPbObD \xrightarrow{4} PabObD \xrightarrow{4} POabD \xrightarrow{4} PODa$



Ex Inversarea intrării

- Ideea: mutarea, **pe rând**, a fiecărui element în poziția corespunzătoare. Mutarea se face prin pași incrementali de interschimbare a elementelor învecinate.

```
1 Reverse(A); A g1, g2;  
2     ag1g2 -> g2ag1;  
3     ag1 -> bg1;  
4     abg1 -> g1a;  
5     a -> .;  
6     -> a;  
7 end
```

- $DOP \xrightarrow{6} aDOP \xrightarrow{2} 0aDP \xrightarrow{2} OPaD \xrightarrow{3} OPbD \xrightarrow{6} aOPbD \xrightarrow{2} PaObD \xrightarrow{3} PbObD$
 $\xrightarrow{6} aPbObD \xrightarrow{3} bPbObD \xrightarrow{6} abPbObD \xrightarrow{4} PabObD \xrightarrow{4} POabD \xrightarrow{4} PODa \xrightarrow{5} POD .$

Aplicații



- “C Language Integrated Production System”;
- Sistem bazat pe **reguli** → “producție” = regulă;
- Principiu de funcționare similar cu al **mașinii Markov**;
- Dezvoltat la NASA în anii 1980;



Exemplu: Minimul a două numere – reprezentare individuală

Ex Exemplu

```
1 (deffacts numbers
2   (number 1)
3   (number 2))
4
5 (defrule min
6   (number ?m)
7   (number ?x)
8   (test (< ?m ?x))
9   =>
10  (assert (min ?m)))
```



- Reprezentarea datelor prin **fapte** → similare simbolurilor mașinii Markov;
- Afirmații despre **atributele** obiectelor;
- Date **simbolice**, construite conform unor **șabloane**;
- Mulțimea de fapte → **baza de cunoștințe** (*factual knowledge base*)

```
1 > (facts)
2 f-0      (initial-fact)
3 f-1      (number 1)
4 f-2      (number 2)
5 For a total of 3 facts.
```



- Similare regulilor mașinii Markov;
- Șablon de **identificare** → secvență de **fapte parametrizate** (vezi variabilele generice ale algoritmilor Markov) și **restricții**;
- Șablon de **acțiune** → secvență acțiuni (assert, retract);
- *Pattern matching* **secvențial** pe faptele din șablonul de identificare;
- **Domeniul de vizibilitate** a unei variabile → restul regulii, după prima apariție a variabilei, în șablonul de identificare.



- Tuplul \langle regulă, fapte asupra cărora este aplicabilă $\rangle \rightarrow$ **înregistrare de activare** (*activation record*);
- Reguli posibil aplicabile asupra diferitelor porțiuni ale **acelorași** fapte;
- Mușimea înregistrărilor de activare \rightarrow **agenda**.

Înregistrări de activare



Exemplu – reluat de mai devreme: minimul a 2 numere

```
1 > (facts)
2 f-0      (initial-fact)
3 f-1      (number 1)
4 f-2      (number 2)
5 For a total of 3 facts.
6
7 > (agenda)
8 0        min: f-1,f-2
9 For a total of 1 activation.
10
11 > (run)
12 FIRE    1 min: f-1,f-2
13 ==> f-3      (min 1)
```



- Principiul refracției:
 - Aplicarea unei reguli o **singură dată** asupra aceluiași fapt și aceluiași porțiuni ale acestora;
 - Altfel, programe care **nu** s-ar termina.
- Terminare:
 - Aplicarea unui număr maxim de reguli \rightarrow (run n);
 - Întâlnirea acțiunii (halt);
 - Golirea agendei.



Ex Exemplu

```
1 (deffacts numbers
2   (numbers 1 2))
3
4 (defrule min
5   (numbers $? ?m $?)
6   (numbers $? ?x $?)
7   (test (< ?m ?x))
8   =>
9   (assert (min ?m)))
```

- Observați utilizarea \$? pentru potrivirea unei secvențe, potențial vidă.



```
1 > (facts)
2 f-0      (initial-fact)
3 f-1      (numbers 1 2)
4 For a total of 2 facts.
5
6 > (agenda)
7 0        min: f-1,f-1
8 For a total of 1 activation.
```

CLIPS – Exemple

Suma oricâtor numere (1)



Exemplu

```
1 (deffacts numbers (numbers 1 2 3 4 5))
2
3 (defrule init
4   ; implicit, (initial-fact)
5   =>
6   (assert (sum 0)))
7
8 (defrule sum
9   ?f <- (sum ?s)
10  (numbers $? ?x $?)
11  =>
12  (retract ?f)
13  (assert (sum (+ ?s ?x))))
```

CLIPS – Exemple

Suma oricâtor numere – Interogare



```
1 > (facts)
2 f-0      (initial-fact)
3 f-1      (numbers 1 2 3 4 5)
4 For a total of 2 facts.
5
6 > (agenda)
7 0        init: *
8 For a total of 1 activation.
9
10 > (run 1)
11 FIRE    1 init: *
12 ==> f-2      (sum 0)
```

CLIPS – Exemple

Suma oricâtor numere – Interogare



```
1 > (agenda)
2 0      sum: f-2,f-1
3 0      sum: f-2,f-1
4 0      sum: f-2,f-1
5 0      sum: f-2,f-1
6 0      sum: f-2,f-1
7 For a total of 5 activations.
8
9 > (run)
10 ciclează!
```




- **Eroarea**: adăugarea unui nou fapt `sum` induce aplicabilitatea repetată a regulii, asupra elementelor **deja** însumate;
- **Corect**: consultarea primului număr din listă și eliminarea acestuia.



Ex Exemplu

```
1 (deffacts numbers (numbers 1 2 3 4 5))
2 (defrule init
3   =>
4     (assert (sum 0)))
5
6 (defrule sum
7   ?f <- (sum ?s)
8   ?g <- (numbers ?x $?rest)
9   =>
10    (retract ?f)
11    (assert (sum (+ ?s ?x)))
12    (retract ?g)
13    (assert (numbers $?rest)))
```

CLIPS – Exemple



Suma oricâtor numere – Interogare pe implementarea corectă

```
1 > (run)
2 FIRE      1  init: *
3 ==> f-2      (sum 0)
4 FIRE      2  sum: f-2,f-1
5 <== f-2      (sum 0)
6 ==> f-3      (sum 1)
7 <== f-1      (numbers 1 2 3 4 5)
8 ==> f-4      (numbers 2 3 4 5)
9 FIRE      3  sum: f-3,f-4
10 <== f-3      (sum 1)
11 ==> f-5      (sum 3)
12 <== f-4      (numbers 2 3 4 5)
13 ==> f-6      (numbers 3 4 5)
```

CLIPS – Exemple



Suma oricâtor numere – Interogare pe implementarea corectă

```
1 FIRE      4 sum: f-5,f-6
2 <== f-5    (sum 3)
3 ==> f-7    (sum 6)
4 <== f-6    (numbers 3 4 5)
5 ==> f-8    (numbers 4 5)
6 FIRE      5 sum: f-7,f-8
7 <== f-7    (sum 6)
8 ==> f-9    (sum 10)
9 <== f-8    (numbers 4 5)
10 ==> f-10  (numbers 5)
11 FIRE     6 sum: f-9,f-10
12 <== f-9    (sum 10)
13 ==> f-11  (sum 15)
14 <== f-10  (numbers 5)
15 ==> f-12  (numbers)
```



Ex Exemplu

```
1 <?xml version="1.0" ?>
2 <persons>
3   <person username="JS1">
4     <name>John</name>
5     <family-name>Smith</family-name>
6   </person>
7   <person username="MI1">
8     <name>Morka</name>
9     <family-name>Ismincius</family-name>
10  </person>
11 </persons>
```

⇓ XSLT ⇓

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <root>
3   <name username="JS1">John</name>
4   <name username="MI1">Morka</name>
5 .</root> . . . . .
```



Transformarea fișierelor XML – Exemplu: sursa

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <xsl:stylesheet xmlns:xsl="http://..." version="1.0">
3   <xsl:output method="xml" indent="yes"/>
4
5   <xsl:template match="/persons">
6     <root>
7       <xsl:apply-templates select="person"/>
8     </root>
9   </xsl:template>
10
11   <xsl:template match="person">
12     <name username="{@username}">
13       <xsl:value-of select="name" />
14     </name>
15   </xsl:template>
16 </xsl:stylesheet>
```



- Ce este și cum funcționează mașina algoritmică Markov: structură, variabile, reguli, algoritmul unității de control.
- Introducere în CLIPS – fapte, reguli, execuție.
- Exemplu de fișier XSLT.

+ | Succes la examen și nu uitați să dați feedback la curs.