Paradigme de Programare

Conf. dr. ing. Andrei Olaru

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

2021

transform from $\rightarrow$ to

then $\rightarrow$.

Cursul 15: Mașina algoritmică Markov

Introducere

Mașina algoritmică Markov

- 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*).

Paradigma asociativă
Caracteristici

- 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

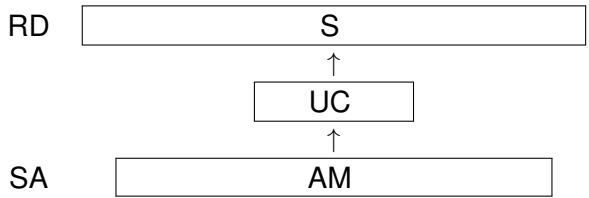
Mașina algoritmică Markov
Exemple de implementare

(implementări fără variabile generice)

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

Structura Mașinii Markov

Perspectivă generală



- 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.

Structura Mașinii Markov

Reguli

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

  `șablon identificare (LHS)` → `șablon substituție (RHS)`

- Exemplu: `a`g_1`c -> ac`
- șabloanele → 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 mașinii (halt).

- De obicei, notate cu `g`, urmat de un indice;
- Mulțimea valorilor pe care le poate lua o variabilă → domeniul variabilei – `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`.

- 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 → variabile generice
- a nedeclarată → variabilă locală ($a \in A_l$)

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

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

+ | Aplicarea regulii

$r : a_1 \ldots a_n \to b_1 \ldots b_m$ asupra unui subșir

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

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

Reguli
Exemplu de aplicare

Exemplu

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

Reguli
Exemplu de aplicare

Exemplu

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

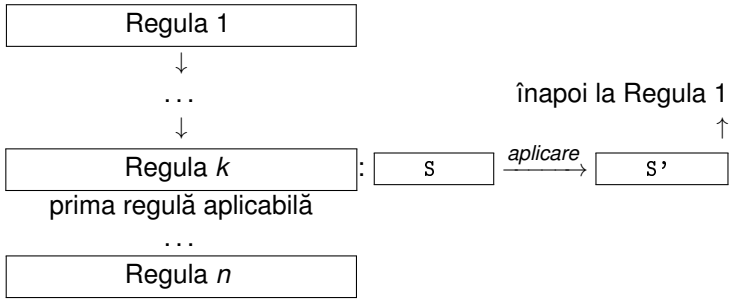
S = 11111 1 2 x 2 y 3 1111
r : 1 g_1 x g_1 y g_2 $\rightarrow$ $1g_2x$
S' = 1111113x1111

Unitatea de control

Aplicabilitate vs. aplicare

- 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

Unitatea de control
Funcționare



Exemplu

Inversarea intrării

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

Exemplu

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

Exemplu

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
```

- $\mathtt{DOP} \xrightarrow{6} \mathtt{aDOP}$

Exemplu

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
```

- $\mathtt{DOP} \xrightarrow{6} \mathtt{aDOP} \xrightarrow{2} \mathtt{OaDP}$

Exemplu

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
```

- $\mathtt{DOP} \xrightarrow{6} \mathtt{aDOP} \xrightarrow{2} \mathtt{OaDP} \xrightarrow{2} \mathtt{OPaD}$

Exemplu

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
```

- $\mathtt{DOP} \xrightarrow{6} \mathtt{aDOP} \xrightarrow{2} \mathtt{OaDP} \xrightarrow{2} \mathtt{OPaD} \xrightarrow{3} \mathtt{OPbD}$

Exemplu

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
```

- $\mathtt{DOP} \xrightarrow{6} \mathtt{aDOP} \xrightarrow{2} \mathtt{OaDP} \xrightarrow{2} \mathtt{OPaD} \xrightarrow{3} \mathtt{OPbD} \xrightarrow{6} \mathtt{aOPbD}$

Exemplu

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
```

- $\texttt{DOP} \xrightarrow{6} \texttt{aDOP} \xrightarrow{2} \texttt{OaDP} \xrightarrow{2} \texttt{OPaD} \xrightarrow{3} \texttt{OPbD} \xrightarrow{6} \texttt{aOPbD} \xrightarrow{2} \texttt{PaObD}$

Exemplu

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
```

- $\mathtt{DOP} \xrightarrow{6} \mathtt{aDOP} \xrightarrow{2} \mathtt{OaDP} \xrightarrow{2} \mathtt{OPaD} \xrightarrow{3} \mathtt{OPbD} \xrightarrow{6} \mathtt{aOPbD} \xrightarrow{2} \mathtt{PaObD} \xrightarrow{3} \mathtt{PbObD}$

Exemplu

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
```

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

Exemplu

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
```

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

Exemplu

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
```

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

Exemplu

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
```

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

Exemplu

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
```

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

Exemplu

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
```

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

Exemplu

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
```

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

Aplicații

CLIPS

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

CLIPS

Exemplu: Minimul a două numere – reprezentare individuală

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.

Înregistrări de activare
Definiție

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

Înregistrări de activare

Exemplu – reluat de mai devreme: minimul a 2 numere

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

> (agenda)
0      min: f-1,f-2
For a total of 1 activation.

> (run)
FIRE    1 min: f-1,f-2
==> f-3     (min 1)
```

Terminarea programelor

- Principiul refracției:
  - Aplicarea unei reguli o singură dată asupra acelorași fapte și acelorași 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.

CLIPS – Exemple

Minimul a două numere – Reprezentare agregată (1)

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.

CLIPS – Exemple

Minimul a două numere – Reprezentare agregată

```
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!
```

CLIPS – Exemple

Suma oricâtor numere – Observații

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

CLIPS – Exemple

Suma oricâtor numere – Implementare corectă

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

```
> (run)
FIRE    1 init: *
==> f-2     (sum 0)
FIRE    2 sum: f-2,f-1
<== f-2     (sum 0)
==> f-3     (sum 1)
<== f-1     (numbers 1 2 3 4 5)
==> f-4     (numbers 2 3 4 5)
FIRE    3 sum: f-3,f-4
<== f-3     (sum 1)
==> f-5     (sum 3)
<== f-4     (numbers 2 3 4 5)
==> 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)
```

XSLT

Transformarea fișierelor XML – Exemplu

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>
```

XSLT

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>
```

Sfârșitul cursului 15

Ce am învățat

- 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.