Paradigme de Programare

S.l. dr. ing. Andrei Olaru
slides: Mihnea Muraru si Andrei Olaru

Catedra de Calculatoare

2013 – 2013, semestrul 2

Cursul 12

Programare asociativă în CLIPS

Cuprins

1. Introducere
2. Fapte şi reguli
3. Exemple
4. Controlul execuţiei

Introducere

CLIPS

- “C Language Integrated Production System”;
- Sistem bazat pe reguli $\rightarrow$ “producţie” = regulă;
- Principiu de funcţionare similar cu al maşinii Markov;
- Dezvoltat la NASA în anii 1980;
- Posibilitatea codificării de implicaţii logice în reguli $\rightarrow$ sisteme expert.

Sisteme expert

Trăsături

- Edward Feigenbaum: “un program inteligent care foloseşte cunoştinţe şi reguli de inferenţă pentru a rezolva probleme suficient de dificile încât să necesite expertiză umană semnificativă”;
- Mimarea procesului de decizie al unui expert uman;
- Limitarea la un domeniu specific $\rightarrow$ dificultatea scrierii unui rezolvitor general de probleme.

Sisteme expert

Aplicaţii

- Configurare de sisteme
- Diagnoză (medicală etc.)
- Educaţie
- Planificare
- Prognoză

...

Fapte şi reguli

Exemplu

Minimul a două numere – reprezentare individuală

Exemplul 42.1.

```
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)))
```

Fapte

- Reprezentarea datelor prin fapte → similare simbolurilor maşinii Markov;
- Afirmaţii despre atributele obiectelor;
- Date simbolice, construite conform unor şabloane;
- Mulţimea de fapte → baza de cunoştinţe (*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.
```

Reguli

- Similare regulilor maşinii Markov;
- Şablon de identificare $\rightarrow$ secvenţă de fapte parametrizate (vezi variabilele generice ale algoritmilor Markov) şi restricţii;
- Şablon de acţiune $\rightarrow$ secvenţă de acţiuni;
- *Pattern matching* secvenţial pe faptele din şablonul de identificare;
- Domeniul de vizibilitate a unei variabile $\rightarrow$ restul regulii, după prima apariţie a variabilei, în şablonul de identificare.

Înregistrări de activare

- Tuplul (regulă, fapte asupra cărora este aplicabilă) $\rightarrow$ înregistrare de activare (*activation record*);
- Reguli posibil aplicabile asupra diferitelor porţiuni ale aceloraşi fapte;
- Muţimea 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)
```

Înregistrări de activare

Principiul refracţiei

- Aplicarea unei reguli o singură dată asupra aceloraşi fapte şi aceloraşi porţiuni ale acestora;
- Altfel, programe care nu s-ar termina.

Terminarea programelor

- Aplicarea unui număr maxim de reguli $\rightarrow$ `(run n)`;
- Întâlnirea acţiunii `(halt)`;
- Golirea agendei.

Exemple

Minimul a două numere

Reprezentare agregată a numerelor – Exemplu

Exemplul 43.1.

```
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)))
```

Minimul a două numere
Reprezentare agregată a numerelor – Observații

- `$?` este o variabilă anonimă, ce se potriveşte cu orice secvenţă, eventual 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.
```

Minimul a două numere

Reprezentare agregată a numerelor – Exemplu (2)

Exemplul 43.2.

```
1  (deffacts numbers (numbers 1 2))
2
3  (defrule min1
4      (numbers ?m ?x)
5      (test (< ?m ?x))
6    =>
7      (assert (min ?m)))
8
9  (defrule min2
10     (numbers ?x ?m)
11     (test (< ?m ?x))
12   =>
13     (assert (min ?m)))
```

Minimul a două numere

Reprezentare agregată a numerelor – Observații (2)

- Selectarea explicită a celor 2 numere împiedică alegerea automată, convenabilă, a acestora, ca în Exemplul 43.1 $\rightarrow$ necesitatea celor 2 reguli.

```
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      min1: f-1
8  For a total of 1 activation.
```

Suma oricâtor numere

Exemplu

Exemplul 43.3.

```
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))))
```

Suma oricâtor numere
Interogare (1)

```
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)
```

Suma oricâtor numere

Interogare (2)

```
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!
```

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 şi eliminarea acestuia.

Exemplul 43.4.

```
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)))
```

Suma oricâtor numere

Interogare pe exemplul corect (1)

```
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)
```

Suma oricâtor numere

Interogare pe exemplul corect (2)

```
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)
```

Accesibilitatea într-un graf

Exemplu de graf

Accesibilitatea într-un graf

Definiția problemei

- Graful: $G = (V, E)$
- Relaţia de accesibilitate: $Acc \subseteq V^2$
- $(u, v) \in E \Rightarrow (u, v) \in Acc$
- $(x, y) \in Acc \land (y, z) \in E \Rightarrow (x, z) \in Acc$

Accesibilitatea într-un graf

Cod (1)

Exemplul 43.5.

```
1  (deftemplate edge (slot from) (slot to) (slot cost))
2  (deftemplate acc (slot source) (slot dest))
3  (deftemplate find (slot source) (slot dest))
4  (deffacts graph
5      (edge (from a) (to b) (cost 1))
6      (edge (from a) (to c) (cost 1))
7      (edge (from a) (to e) (cost 9))
8      (edge (from b) (to c) (cost 2))
9      (edge (from b) (to d) (cost 1))
10     (edge (from c) (to a) (cost 2))
11     (edge (from c) (to d) (cost 1))
12     (edge (from c) (to e) (cost 4))
13     (edge (from d) (to e) (cost 2))
14     (find (source a) (dest e)))
```

Accesibilitatea într-un graf

Cod (2)

Exemplul 43.5 (continuare).

```
16 (defrule base
17     (edge (from ?x) (to ?y))
18   =>
19     (assert (acc (source ?x) (dest ?y))))
20
21 (defrule expand
22     (acc (source ?x) (dest ?y))
23     (edge (from ?y) (to ?z))
24   =>
25     (assert (acc (source ?x) (dest ?z))))
```

Accesibilitatea într-un graf

Cod (3)

Exemplul 43.5 (continuare).

```
27 (defrule found
28     (find (source ?x) (dest ?y))
29     (acc (source ?x) (dest ?y))
30   =>
31     (printout t "Found" crlf)
32     (halt)) ; Ne oprim cand raspundem afirmativ.
```

Controlul execuţiei

- Exemplul 43.5: posibilitatea continuării explorării grafului după obţinerea răspunsului căutat;
- Optimizare: forţarea aplicării regulii `found`, imediat după identificarea răspunsului;
- Problemă: aplicabilitatea concomitentă a regulilor `expand` şi `found`;
- Soluţie: prioritizarea regulii `found`.

Exemplul 44.1 (optimizare a exemplului 43.5).

```
1 (defrule found
2     (declare (salience 10))
3     (find (source ?x) (dest ?y))
4     (acc (source ?x) (dest ?y))
5   =>
6     (printout t "Found" crlf)
7     (halt)) ; Ne oprim cand raspundem afirmativ.
```

Salience

- *Salience* = prioritatea în aplicare a unei reguli;
- Implicit 0, posibil negativă;
- Valoare mai mare → prioritate mai mare.

Minimul oricâtor numere

Determinare iterativă

Exemplul 44.2.

```
1  (deffacts numbers (numbers 5 7 1 3))
2
3  (defrule init
4      (not (min ?m))
5      (numbers ?x $?rest)
6    =>
7      (assert (min ?x)))
```

Exemplul 44.2.

```
 9  (defrule compute
10      ?f <- (min ?m)
11      (numbers $? ?x $?)
12      (test (< ?x ?m))
13    =>
14      (retract ?f)
15      (assert (min ?x)))
16
17  (defrule print
18      (declare (salience -10)) ; compute neaplicabila
19      (min ?m)
20    =>
21      (printout t ?m crlf))
```

Exemplul 44.3.

```
1 (deffacts numbers (numbers 1 5 7 3))
2
3 (defrule min
4     (numbers $? ?m $?)
5     (not (numbers $? ?x & :(< ?x ?m) $?))
6   =>
7     (assert (min ?m))
8     (printout t ?m crlf))
```

- Definirea de condiţii *inline* asupra variabilelor, prin `&`;
- Citirea regulii: “Minimul este acel element pentru care nu găsim altul mai mic”.

Drumurile optime în graf

Pe exemplul anterior de graf

Drumurile optime într-un graf cu costuri

Principiu

- Drumurile optime $source \rightsquigarrow dest$ sunt drumurile *utile* $source \rightsquigarrow dest$, în momentul în care nu se mai pot obţine alte drumuri *utile* prin extinderea drumurilor *utile* existente;
- Iniţial, există un singur drum, ce conţine doar nodul *source* şi are costul 0;
- Un drum $x \rightsquigarrow y, z$ extinde un drum $x \rightsquigarrow y$ dacă
  $(y,z) \in E$ şi $z \notin x \rightsquigarrow y$, unde
  $cost(x \rightsquigarrow y, z) = cost(x \rightsquigarrow y) + cost(y,z)$;
- Un drum $x \rightsquigarrow y$ se numeşte *util* dacă nu există un alt drum $x \rightsquigarrow y$ mai ieftin. Drumurile *neutile* sunt imediat eliminate în timpul explorării.

Drumurile optime într-un graf cu costuri

Implementare (1)

Exemplul 44.4.

```
1 (deftemplate edge (slot from) (slot to) (slot cost))
2 (deftemplate find (slot source) (slot dest))
3 (deftemplate path (multislot nodes) (slot cost))
4
5 (defrule init
6       (find (source ?s))
7    =>
8       (assert (path (nodes ?s) (cost 0))))
```

Drumurile optime într-un graf cu costuri

Implementare (2)

Exemplul 44.4 (continuare).

```
10 (defrule expand
11     (path (nodes $?prefix ?last) (cost ?pc))
12     (edge (from ?last) (to ?neighbor) (cost ?ec))
13     (test (and  (neq ?neighbor ?last)
14                 (not (member ?neighbor $?prefix))))
15   =>
16     (assert (path   (nodes $?prefix ?last ?neighbor)
17                     (cost (+ ?pc ?ec)))))
```

Exemplul 44.4 (continuare).

```
19 (defrule prune
20     (declare (salience 10))
21     (path (nodes $? ?dest) (cost ?gc))
22     ?f <- (path (nodes $? ?dest) (cost ?bc))
23     (test (> ?bc ?gc))
24   =>
25     (retract ?f))
26
27 (defrule announce
28     (declare (salience -10))
29     (find (dest ?d))
30     (path (nodes $?prefix ?d))
31   =>
32     (printout t $?prefix " " ?d crlf))
```

Drumurile optime într-un graf fără costuri

Problema

- Criteriul optimizat: numărul de muchii;
- Soluţia 1: abordarea precedentă, presupunând că toate muchiile au costul 1;
- Soluţia 2: parcurgere în lăţime.

Drumurile optime într-un graf fără costuri

Strategia CLIPS pe lățime

- Parcurgere în lăţime $\rightarrow$ necesitatea extinderii, într-un pas, a unei cele mai scurte căi;
- Observaţie: vârsta superioară a faptelor reprezentând căi mai scurte;
- Soluţie: alterarea ordinii în care faptele sunt evaluate în raport cu şabloanele de identificare ale regulilor.

Drumurile optime într-un graf fără costuri

Implementare (1)

Exemplul 44.5.

```
1  (deftemplate edge (slot from) (slot to) (slot cost))
2  (deftemplate find (slot source) (slot dest))
3
4  (defrule init
5      (find (source ?s))
6    =>
7      (assert (path ?s))
8      (set-strategy breadth))
```

Drumurile optime într-un graf fără costuri

Implementare (2)

Exemplul 44.5 (continuare).

```
10 (defrule expand
11     (path $?prefix ?last)
12     (edge (from ?last) (to ?neighbor))
13     (test (and  (neq ?neighbor ?last)
14                 (not (member ?neighbor $?prefix))))
15   =>
16     (assert (path $?prefix ?last ?neighbor)))
17
18 (defrule announce
19     (declare (salience 10))
20     (find (dest ?d))
21     (path $?prefix ?d)
22   =>
23     (printout t $?prefix ?d crlf) (halt))
```

Strategii

Pentru ordinea potrivirii faptelor în CLIPS

- Ordinea în care faptele sunt evaluate în raport cu şabloanele de identificare ale regulilor;
- *Depth* (implicită): cel mai recent fapt potrivit primul;
- *Breadth*: cel mai vechi fapt potrivit primul;
- *Random*: alegere aleatorie.

Sfârșitul cursului 12

Ce am învățat

· CLIPS: fapte și reguli, înregistrări de activare, exemple de utilizare pe probleme simple, elemente de controlul execuției – salience și strategii.