Documentatie laborator 9 – Programare asociativa in CLIPS

CLIPS

CLIPS este un limbaj bazat pe reguli de tip „daca-atunci” (ex: daca e frig, atunci imi iau fular si manusi). Il puteti downloada din pagina cursului. Pentru a rula un program CLIPS scrieti continutul sau intr-un fisier .clp, apoi deschideti CLIPS si, in meniul acestuia, alegeti pe rand:

- Load (pe fisierul cu programul)
- Reset
- Run

Pentru a observa efectul programului, bifati in meniu Execution->Watch->Facts. Aceasta optiune va permite sa urmariti in timpul rularii ce fapte intra si ies din baza de fapte, ca efect al aplicarii diverselor reguli.

Pentru un tutorial detaliat consultati
http://clipsrules.sourceforge.net/documentation/v630/bpg.pdf

Sintaxa

- Notatie prefixata si constructori inchisi intre paranteze (ca in Scheme)
  ex: (+ 2 4 5)
- Case sensitive
- Comentariile sunt precedate de ; (ca in Scheme)
- Exista o notatie speciala pentru variabile
  - ? – variabila anonima
  - ?x – variabila cu numele x
  - $? – variabila multicamp anonima
  - $?x – variabila multicamp cu numele x
    (o variabila multicamp este o variabila care poate face pattern match cu 0, 1 sau mai multe valori, nu neaparat valori de acelasi tip)

Exemplu de cod continand variabile:

```
(deffacts stuff (fact 1 2 3) (fact 4 5) (gossip 4 a 6 b))
(defrule test
        (fact $?a $?b)
=>
        (printout t $?a $?b crlf))

; se definesc 3 fapte
```

; atentie: fact NU este un identificator pt (1 2 3) sau pt (4 5)
; se defineste o regula test care face pattern match pe faptele
; de "fact si inca ceva"
; posibile pattern match-uri:
; in (fact 1 2 3), $?a se leaga la un sir vid si $?b la 1 2 3
; in (fact 1 2 3), $?a se leaga la 1 si $?b la 2 3
; ...
; in (fact 1 2 3), $?a se leaga la 1 2 3 si $?b la un sir vid
; in (fapt 4 5) $?a nimic si $?b 4 5, $?a 4 si $?b 5, $?a 4 5 si $?b nimic

; deffacts este un constructor cu care definim un numar de fapte
; el trebuie sa poarte un nume (aici stuff), care nu e un identificator pt acele fapte

; la fel defrule e un constructor pt o regula
; numele sau NU inseamna ca regula respectiva poate fi invocata ca o procedura!

; atentie: nu are rost sa incercati sa afisati continutul variabilei gossip
; este o greseala frecventa!
; gossip nu pointeaza la 4 a 6 b
; pot exista mai multe fapte (gossip 1 3) (gossip mere pere)
; caz in care daca am incerca sa facem ceva cu gossip pe post de variabila
; nu am sti la care din faptele de mai sus ne referim

Fapte si template-uri

In CLIPS, universul problemei este reprezentat prin fapte. Totalitatea faptelor formeaza baza de fapte a programului, al carei continut variaza pe parcursul executiei. De exemplu, un fapt din universul unei probleme pe grafuri ar putea fi (muchie (sursa a) (dest b) (cost 4)).

Un template este o inregistrare care contine campuri/sloturi (cam ca un struct in C). Template-urile trebuie declarate explicit in program daca nu se reduc la un singur multislot. Exemple de (definire de) template-uri:

; template-ul pt faptul muchie de mai sus
(deftemplate muchie (slot sursa) (slot destinatie) (slot cost))
; multislot inseamna ca sloul respectiv poate contine 0, 1 sau mai multe valori
(deftemplate cale (multislot noduri) (slot cost))

Operatii uzuale asupra faptelor

- assert (introduce un fapt in baza de fapte)
- retract (retrage un fapt din baza de fapte)
- modify (modifica un fapt existent in baza de fapte)
- duplicate (duplica un fapt din baza de fapte)

Reguli

Un algoritm CLIPS este codificat prin reguli. Acestea au nume unice si nu functioneaza ca niste proceduri (nu pot fi aplicate cand doreste utilizatorul, ci conform mecanismului explicat ceva mai jos). Sintaxa unei reguli este urmatoarea:

(defrule nume
pattern1
...
patternn
=>
actiune1
...
actiunem)

Un pattern reprezinta un fapt parametrizat (poate contine variabile ca valori ale sloturilor), eventual incomplet (nu neaparat toate sloturile sunt precizate, ci numai cele de interes pentru regula curenta). Pentru a testa daca o regula este aplicabila, mecanismul este urmatorul:

- se cauta un fapt in baza de fapte care sa se potriveasca cu pattern1
- un fapt se potriveste cu un pattern daca exista o instantiere a variabilelor din pattern la valori a.i. patternul sa ajunga sa coincida cu faptul (in urma acestei instantieri)
- odata instantiate variabilele dintr-un pattern la valori, aceste instantieri vor fi recunoscute (raman valabile) in restul constructorului defrule
- se trece la gasirea unui fapt care sa se potriveasca cu patternul urmator
- regula este aplicabila daca s-a reusit potrivirea tuturor patternurilor (dupa ce s-au incercat toate combinatiile de fapte posibile)
- cand se aplica o regula, actiunile ei sunt executate secvential

Exemplu de pattern:

(muchie (sursa a) (dest ?d))

Acest pattern se potriveste cu faptul *(muchie (sursa a) (dest b) (cost 2))*, daca legam variabila ?d la valoarea b.

Prioritati asociate regulilor

Regulilor li se pot asigna prioritati astfel incat unele sa se aplice cu prioritate in fata altora, atunci cand mai multe reguli sunt aplicabile.

Exemplu:

```
(defrule test-prio
        (declare (salience 10))
=>
        (assert (program-inceput)))
```

Cu cat salience-ul este mai mare, cu atat regula are prioritate mai mare.

Fluxul de executie

- motorul de inferenta cauta reguli aplicabile in baza de reguli
- cand sunt mai multe reguli aplicabile, ele se ordoneaza dupa salience; intre regulile cu aceeasi prioritate se alege una la intamplare
- se aplica regula aleasa
- procesul se repeta pana cand nu mai exista reguli aplicabile

Principiul refractiei

O regula se poate aplica o singura data pentru un acelasi set de fapte cu un acelasi set de legari ale variabilelor din patternuri. Acest principiu impiedica adesea aplicarea aceleiasi reguli la infinit.

Exemple de programe CLIPS

```
;; maximul dintre mai multe numere, varianta 1
(deffacts fapte (numar 1) (numar 3) (numar 2) (maxim -999))

(defrule maxim
        (numar ?x)                  ; daca gasim un numar
        ?f<-(maxim ?y&:(> ?x ?y))   ; si maximul curent e mai mic ca el
=>
        (retract ?f)                ; retragem faptul care tine maximul curent
        (assert (maxim ?x))         ; introducem un nou fapt care tine noul maxim
)
; ?y&:(> ?x ?y) reprezinta o constructie care impune asupra variabilei ?y o conditie
; aici ea este ?x>?y, in particular pot fi diverse alte conditii
; ?f<- reprezinta o constructie care preia indicele faptului urmator intr-o variabila ?f
; acest indice este un identificator unic pt fapt si il putem folosi pt a retrage faptul

;; maximul dintre mai multe numere, varianta 2 (toate numerele sunt tinute intr-un fapt sir)
(deffacts fapte (sir 3 2 5 3 4 6 1 2))

(defrule maxim
        (sir $? ?x $?)
```

```
        (not (sir $? ?y&:(> ?y ?x) $?)) ; nu exista un fapt sir care sa contina un ?y > ?x
=>
        (assert (maxim ?x))
)

;; varianta 3:  rezulta ca si in primul caz am fi putut scrie
(defrule maxim
        (numar ?x)
        (not (numar ?y&:(> ?y ?x)))
=>
        (assert (maxim ?x))
)
```