SE-Laborator9

CLIPS C Language Integrated Production System

COOL CLIPS Object-Oriented Language

CLIPS Rule-base programming language

COOL Object oriented extension to CLIPS

Definirea unei clase: sloturi si a unei relatii IS-A

```
(defclass ship                                  ;defineste o clasa cu numele ship
        (is-a INITIAL-OBJECT)                   ;system-class INITIAL-OBJECT
        (slot x-velocity (create-accessor read-write))   ;sloturi cu drept de r/w
        (slot x-velocity (create-accessor read-write))
); de la defclass ship

;definirea instantei pentru o clasa:

(definstances ships                             ;definirea unui set de instante ships
        (titanic of ship)                       ;definim titanicul ca nava
                                                ;       cu setarea valorilor pentru sloturi
        (x-velocity 10)
        (y-velocity 12)
); de la definstances ships
```

CLIPS/COOL - Remarci asupra sintaxei asemanatoare cu LISP

LISP este bazat pe liste, secvente de atomi sau structuri puse intr-o lista, delimitata de paranteze:

(S_1 .... S_n)

Listele pot contine la randul lor alte liste.

LISP este un limbaj de programare functional. Programul consista in mare dintr-un set de definitii de functii. Rularea unui program consta din evaluarea unor functii specificate.

```
(deffun <function-name> (<parameters>)
        (<body>)
); de la deffun
```

In CLIPS/COOL, elementele de limbaj precum defclass, defmethod, slot, ...sunt de fiecare data primul argument al unei liste.

Functii de baza in CLIPS/COOL

Definirea unei clase:

```
(defclass <class-name>
        (is-a <super-class>)                                    ; de obicei clasa system USER
        (role <abstract sau concrete>)                          ; clasele abstracte nu pot avea instante
        ...
        (slot <slot-name> (type <slot-type>) <slot-specifications>))
                                                                ;slot-type defineste structura de data pentru
                                                                ;       slot (de exemplu INTEGER)
                                                                ;slot-specifications defineste constrangerile
                                                                ;       si caracteristicile speciale pentru
                                                                ;       slot, de exemplu access-type si
                                                                ;       default-value
```

Definirea metodelor

```
(definstances ships
        (titanic of ship)
        (x-velocity 10)
        (y-velocity 12)
); de la definstances ships
```

Definirea unei metode pentru o clasa:

```
(defmessage-handler ship calc-speed ( )                         ; fara parametrii
        ;urmeaza functia pentru calcularea vitezei
        (sqrt   (+      (* ?self:x-velocity ?self:x-velocity)
                        (* ?self:x-velocity ?self:x-velocity)
                )
        )
)
```

Se defineste metoda calc-speed pentru clasa ship, folosind valorile sloturilor ale obiectului propriu-zis (care primeste mesajul). ?self este obiectul respectiv, : este selectorul, x-velocity reprezinta slotul selectat.

Transmiterea de mesaje:

```
(send [object] message-name)                    ;obiectul este o instanta generala
```

examplu: (send [titanic] calc-speed)

transmite mesajul calc-speed obiectului titanic. Instanta/obiect titanic foloseste metoda mostenita calc-speed (de la ship) pentru a calcula viteza, pe baza valorilor propriilor sloturi.

Intoarce ca raspuns la acest mesaj valoarea calculata pentru viteza, rezultatul evaluarii calc-speed pentru titanic.

```
(defclass person
        (is-a USER)             ;system-class USER
)

(defclass quaker
```

```
        (is-a person)
)

(defclass republican
        (is-a person)
)

(defclass republican-quaker
        (is-a republican quaker)
        (role concrete)
)
```

```
; Jocul vietii
; o celula poate avea maxim 8 vecini si in fiecare stadiu al jocului, starea ei este guvernata de urmatoarele ;
; reguli:
;       a. O celula care este vie si are mai putin de 3 sau mai mult de 4 celule vii ca vecini, moare
;       b. O celula moarta cu 3 sau 4 vecini vii, reinvie


(defmodule MAIN (export ?ALL))
(deftemplate generation (slot gen) (slot changes))

(defclass cell (is-a USER)
        (role concrete) (pattern-match reactive)
        (slot x (create-accessor read-write)
                (pattern-match non-reactive))
        (slot y (create-accessor read-write)
                (pattern-match non-reactive))
        (slot generation (default 1))
        (slot state (create-accessor write) (default alive))
        (slot live-neighbors (default 0))
)

(deffacts fff (max_generation 30)
        (generation (gen 0) (changes 1))
)

(definstances ff
([c01] of cell (x 0) (y 1) (state dead))
([c03] of cell (x 0) (y 3))
([c10] of cell (x 1) (y 0))
([c11] of cell (x 1) (y 1) (state dead))
([c12] of cell (x 1) (y 2))
([c13] of cell (x 1) (y 3) (state dead))
([c14] of cell (x 1) (y 4))
([c21] of cell (x 2) (y 1))
([c23] of cell (x 2) (y 3))
([c30] of cell (x 3) (y 0))
([c32] of cell (x 3) (y 2))
([c33] of cell (x 3) (y 3))
([c34] of cell (x 3) (y 4))
([c41] of cell (x 4) (y 1))
([c42] of cell (x 4) (y 2))
([c43] of cell (x 4) (y 3))
([c55] of cell (x 5) (y 5)))
```

```
(defmessage-handler cell turn (?how)
        (bind ?self:state ?how)
        (bind ?self:generation (+ ?self:generation 1))
)

(defmessage-handler cell is_neighbor (?how)
        (bind ?self:live-neighbor
                (if (eq ?how dead) then (- ?self:live-neighbors 1)
                        (else (+ ?self:live-neighbors 1)))
)

(defmessage-handler cell neighbor (?ob)
        (and (<= (abs (- ?self:x (send ?ob get-x))) 1)
                (<= (abs (- ?self:y (send ?ob get-y))) 1))
)

(defrule init-live-neighbors
        (generation (get 0))
        ?l <- (object (is-a cell) (name ?x) state alive))
        ?ob <- (object (is-a cell) (name ~?x))
        (test (send ?ob neighbor ?l)
)
=>
(send ?ob_neighbor alive))

(defrule cell_dies
        ?f <- (generation (gen ?g) (changes ?c))
        ?ob <- (object (is-a cell) (name ?id)
        (state alive)
        (generation ?gg&:(<= ?gg ?g))
        (live-neighbors ?n))
        (test (or (< ?n 3) (> ?n 4)))
=>
        (printout t “cell” ?id “ dies” crlf)
        (modify ?f (changes (+ ?c 1)))
        (send ?ob turn dead)
)

(deffrule cell_resurects
        ?f <- (generation (gen ?g) (changes ?c))
        ?ob <- (object (is-a cell) (name ?id)
        (state dead)
        (generation ?gg&:(<= ?gg ?g))
        (live-neighbors ?n))
        (test (or (= ?n 3) (= ?n 4)))
=>
        (printout t “cell” ?id “ resurects” crlf)
        (modify ?f (changes (+ ?c 1)))
        (send ?ob turn alive)
)

(defrule next_generation
        (declare (salience –10))
=>
        (refresh next_generation)
```

```
        (focus NextGen)
)

(defmodule NextGen (import MAIN ?ALL))

(defrule modify_context
        (declare (salience 10))
        ?d <- (object (is-a cell) (name ?x) (state ?how))
        ?ob <- (object (is-a cell) (name ~?x))
        (test (send ?ob neighbor ?d))
=>
        (send ?ob is_neighbor ?how)
)

(defrule next_generation
        (max_generation ?max)
        ?f<-(generation (gen ?g) (changes ?c))
=>
        (if (or (>= ?g ?max) (= ?c 0)) then (halt)
        else
                (modify ?f (gen (+ ?g 1)) (changes 0))
                (printout t "==== generation: " (+ ?g 1) " ====" crlf)
                (return)
        )
)
```