CLIPS 2 – Variables & rules matching facts

Παρασκευή, 3 Απριλίου 2020 9:37 πμ

Clips σε Ubuntu ≥18 (ή Windows Linux Subsystem)

Δείτε πως να εγκαταστήσετε κάνοντας compile το source code εδώ:

https://asciinema.org/a/lyLJTg7wv3FRGr4nz7ZdfpDys

Προσοχή: στο σημείο που κάνουμε compile τον πηγαίο κώδικα της clips η εντολή δεν φαίνεται σωστά, κρύβεται ένα m στο τέλος, η ορθή εντολή είναι:

```
gcc -DGENERIC=1 -DEMACS_EDITOR=0 -o clips *.c -lm
```

Θέλουμε να κάνουμε κάτι με τις ημέρες της εβδομάδας, ας τις ορίσουμε:

```
(assert (day Monday))
(assert (day Tuesday))
(assert (day Tuesday))
```

..κ.ο.κ..

ή εναλλακτικά τα δηλώνουμε ως persistent facts τα οποία *ξαναφορτώνονται* κάθε φορά που η clips γίνεται (reset):

```
(deffacts days-of-the-week
    (day Monday) (day Tuesday) (day Wednesday)
    (day Thursday) (day Friday) (day Saturday) (day Sunday) )

(facts)
```

; πού είναι;

`(reset)` ;προσοχή!! *όχι (clear)*, θα τα σβήσει όλα

==========

`(clear)` ;τα σβήνουμε όλα και πάμε από αρχή.

```
(deffacts days-of-the-week
    (day Monday) (day Tuesday) (day Wednesday)
    (day Thursday) (day Friday) (day Saturday) (day Sunday) )
```

; ταυτοποίηση με ?

; ας ορίσουμε ένα κανόνα

```
(defrule find-day
  (day ?)
  =>
  (printout t "A day of the week was found" crlf))
```

```
(agenda)
```
;ας ελέγξουμε ποιοι κανόνες ενεργοποιήθηκαν..

;γιατί δεν ενεργοποιείται κανένας κανόνας????..... :-/

```
(reset)
```
;**μόνο τότε τρέχει** το deffacts, μετά το τελευταίο (clear) !!

```
(agenda)
```
;...να η ενεργοποίηση των κανόνων

```
(run)
```
;… και η εκτέλεσή τους

; ωραία, αλλά καλύτερα να τυπώναμε το όνομα κάθε μέρας..

```
(defrule find-day
  (day ?name)
  =>
  (printout t "Day of the week " ?name " was found" crlf))
```

; τσεκάρουμε agenda

```
CLIPS> (agenda)
0      find-day: f-7
0      find-day: f-6
0      find-day: f-5
0      find-day: f-4
0      find-day: f-3
0      find-day: f-2
0      find-day: f-1
For a total of 7 activations.
```

; ας προσθέσουμε μια ακόμη ημέρα, γιορτινή, την Καθαρά Δευτέρα

```
(assert (day Clean Monday))
```

; τσεκάρουμε facts

```
(facts)
f-0     (initial-fact)
f-1     (day Monday)
f-2     (day Tuesday)
f-3     (day Wednesday)
f-4     (day Thursday)
f-5     (day Friday)
f-6     (day Saturday)
f-7     (day Sunday)
f-8     (day Clean Monday)
```

; τσεκάρουμε πάλι agenda

```
(agenda)
0      find-day: f-7
0      find-day: f-6
0      find-day: f-5
0      find-day: f-4
0      find-day: f-3
0      find-day: f-2
0      find-day: f-1
For a total of 7 activations.
```

; γιατί πάλι 7 activations, όχι 8; γιατί το f-8 δεν είναι στα activations
; ποιο pattern αναζητά η find-day? `(day ?name)`
; τι μορφή έχει το f-8? `(day Clean Monday)`

==========

; ορίζουμε όλα τα facts του ζωόκοσμου:

```
(deffacts animal-life
     (animal dog) (animal cat) (animal duck)
     (animal turtle) (warm-blooded dog) (warm-blooded cat)
     (warm-blooded duck) (lays-eggs duck) (lays-eggs turtle)
     (child-of dog puppy) (child-of cat kitten)
     (child-of turtle hatchling)
)
```

; reset για να προστεθούν στη βάση γνώσης

```
(reset)
```

; ο κανόνας mamal κάνει assert ένα νέο γεγονός
; η μεταβλητή ?name έχει εμβέλεια εντός του κανόνα, και στο αριστερό και στο δεξί μέλος του.
; η μεταβλητή ?name θα πρέπει να εμφανίζεται με την ίδια τιμή σε όλα τα facts που επιχειρείται να γίνουν match στο LHS (αριστερό μέρος) του κανόνα
; τέλος, στα προαπαιτούμενα για την ενεργοποίηση του κανόνα μπορεί να

μετέχουν και λογικοί τελεστές, πχ not. Κάθε προαπαιτούμενο, αν δεν ορίζεται αλλιώς, συνδέεται με λογικό and με όλα τα άλλα, δλδ εδώ πρέπει να υπάρχει *και* ένα fact (animal xxx) *και* ένα fact (warm-blooded xxx) *και* να μην υπάρχει fact (lays-eggs xxx).

```
(defrule mammal
  (animal ?name)
  (warm-blooded ?name)
  (not (lays-eggs ?name))
  =>
  (assert (mammal ?name))
  (printout t ?name " is a mammal" crlf))
```

; το νέο γεγονός είναι ακριβώς σαν να το κάναμε εμείς assert

```
(run)

(facts)
f-0     (initial-fact)
f-1     (day Monday)
f-2     (day Tuesday)
f-3     (day Wednesday)
f-4     (day Thursday)
f-5     (day Friday)
f-6     (day Saturday)
f-7     (day Sunday)
f-8     (animal dog)
f-9     (animal cat)
f-10    (animal duck)
f-11    (animal turtle)
f-12    (warm-blooded dog)
f-13    (warm-blooded cat)
f-14    (warm-blooded duck)
f-15    (lays-eggs duck)
f-16    (lays-eggs turtle)
f-17    (child-of dog puppy)
f-18    (child-of cat kitten)
f-19    (child-of turtle hatchling)
f-20    (mammal cat)
f-21    (mammal dog)
```

; ένας ακόμη κανόνας που κάνει νέα assert
; σε αυτόν το δεύτερο προαπαιτούμενο χρησιμοποιεί δύο μεταβλητές,
η δύτερη μεταβλητή χρησιμοποιείται για το assert στο RHS.

```
(defrule mammal2
  (mammal ?name)
  (child-of ?name ?young)
  =>
  (assert (mammal ?young))
  (printout t ?young " is a mammal" crlf))

(run)
==> f-22    (mammal puppy)      ; γίνεται assert ένα ακόμη fact
```

```
puppy is a mammal
==> f-23    (mammal kitten)
kitten is a mammal
```

; γίνεται assert ένα ακόμη fact

; ένας κανόνας εκτός από assert μπορεί και να κάνει retract, να αφαιρεί γεγονότα από τη βάση γνώσης

; για να το πετύχει αυτό προσέξτε πώς στο LHS παίρνει μια αναφορά προς το γεγονός που θα αφαιρεθεί, όχι προς το περιεχόμενό του

προσοχή: στη διάκριση ?d και ?dname

```
(defrule rem-days
    ?d <- (day ?dname)
    =>
    (printout t ?dname crlf) (retract ?d))
```

; τέλος, ένας κανόνας μπορεί να κάνει κάνει read μια τιμή από το χρήστη ώστε να συλλέξει τις τιμές που χρειάζεται για να κάνει πχ assert ένα νέο fact

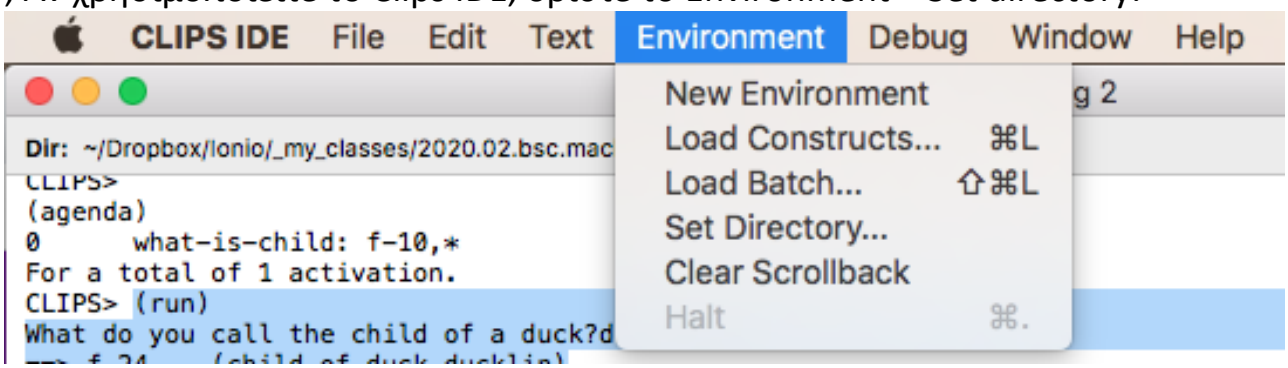
```
(defrule what-is-child
        (animal ?name)
        (not (child-of ?name ?))
     =>
        (printout t "What do you call the child of a " ?name "?")
        (assert (child-of ?name (read))))

(run)
What do you call the child of a duck?ducklin
==> f-24    (child-of duck ducklin)
```

==========

; Μπορούμε να φορτώνουμε κώδικα clips ο οποίος είναι αποθηκευμένος σε αρχείο .clp.

; Αν χρησιμοποιείτε το Clips IDE, ορίστε το Environment > Set directory:



; στη συνέχεια φορτώστε το αρχείο .clp που θέλετε με χρήση της load

```
(load Solution_1.1.clp)
Defining defrule: call-the-aa +j+j+j
Defining defrule: buy-some-petrol +j+j+j
Defining defrule: replace-starter +j+j+j+j+j
```

```
Defining defrule: clean-terminals +j+j+j+j+j
Defining defrule: replace-solenoid +j+j+j+j+j
Defining defrule: replace-fuse +j+j+j+j+j
Defining defrule: charge-battery +j+j+j
Defining defrule: are-lights-working +j+j
TRUE

(rules)
call-the-aa
buy-some-petrol
replace-starter
clean-terminals
replace-solenoid
replace-fuse
charge-battery
are-lights-working
For a total of 8 defrules.

(agenda)
0      are-lights-working: *      ; ένας κανόνας ήδη έχει ενεργοποιηθεί
For a total of 1 activation.

(ppdefrule are-lights-working)    ; ας δούμε τον ορισμό του
(defrule MAIN::are-lights-working
   (not (lights-working ?))
   =>
   (printout t "Are the car's lights working (yes or no)?")
   (assert (lights-working (read))))    ; προσέξτε την είσοδο χρήστη

(run)
Are the car's lights working (yes or no)?yes
==> f-1     (lights-working yes)
```

Άσκηση: ορίστε τους κανόνες που απαιτούνται για να υλοποιήσετε ολόκληρο το διαγνωστικό σύστημα της περασμένης εβδομάδας.

==========

Think deeper: Έστω ένα set κανόνων που έχει υλοποιήσει το πιο ζητούμενο. Ας το τρέξουμε;

```
(run)
Are the terminals clean (yes or no)?yes
==> f-1     (terminals-clean yes)
Is the solenoid fuse ok (yes or no)?no
==> f-2     (solenoid-fuse-ok no)
Can you hear the solenoid click (yes or no)?yes
==> f-3     (solenoid-click yes)
Is there any petrol in the tank (yes or no)?no
==> f-4     (petrol no)
Is the car's starter turning (yes or no)?yes
==> f-5     (starter-turning yes)
Buy some petrol
```

```
Are the car's lights working (yes or no)?yes
==> f-6     (lights-working yes)
```

Μετά το assert του f-5, το διαγνωστικό ήδη μας απαντά τι πρέπει να κάνουμε, γιατί λοιπόν συνεχίζει και με την ερώτηση "`Are the car's lights working (yes or no)?`";
Επίσης, η σειρά των ερωτήσεων, είναι ξ καλύτερη δυνατή; Πχ ακολουθεί την top-down προσέγγιση του διαγράμματος; Πώς μπορείτε να πετύχετε κάτι τέτοιο;