

CLIPS

Σύντομη Εισαγωγή -
Περιγραφή του Μηχανισμού
Εκτέλεσης

Ιστορία της CLIPS

- CLIPS = C Language Integrated Production System
- Αναπτύχθηκε στη NASA τη δεκαετία του 1980
- Η γλώσσα υλοποίησης είναι η C
- Υποστηρίζει τρεις τύπου προγραμματισμού:
 - Βασισμένο σε κανόνες (rule-based)
 - Διαδικαστικό
 - Αντικειμενοστρεφή
- Επεκτάσεις/Παραλλαγές
 - COOL: CLIPS Object-Oriented Language
 - **JESS: Java Expert Systems Shell**
 - Fuzzy Clips: Ενσωμάτωση ασαφούς λογικής (fuzzy logic)

CLIPS

- Περιβάλλον Προγραμματισμού με βάση τους κανόνες (rule-based)
- Βασικά στοιχεία: **γεγονότα (facts), κανόνες (rules)**
- Βασικές συνιστώσες: λίστα γεγονότων, βάση γνώσης (κανόνες), μηχανισμός εξαγωγής συμπερασμάτων
- Αιτιολόγηση προς τα εμπρός (forward chaining)
- <http://clipsrules.sourceforge.net/>

Το περιβάλλον CLIPS

- Όταν ξεκινάμε το περιβάλλον της CLIPS εμφανίζεται το ακόλουθο prompt:
 - `CLIPS>`
στο οποίο γράφουμε τις εντολές της CLIPS
- Η CLIPS βασίζεται σε διερμηνέα (interpreter)
- Όλες οι εντολές σε παρενθέσεις π.χ. (exit)

Γεγονότα (Facts)

- Η CLIPS βασίζεται στην αξιοποίηση και διαχείριση των γεγονότων
- Τα γεγονότα διατηρούνται στη λίστα γεγονότων (fact list)
- Τα γεγονότα αποτελούνται από i) συμβολικό όνομα, ii) συμβολικά πεδία (**slots**) με τις τιμές τους
- Παράδειγμα:
 - (person (name 'John') (age 23) (height 1.87))
- Η σειρά που αναγράφονται τα συμβολικά πεδία δεν παίζει ρόλο.
- Όταν υπάρχει μοναδικό πεδίο με όνομα ίδιο με το όνομα του γεγονότος, τότε το πεδίο παραλείπεται π.χ. αντί (num (num 50)) μπορούμε να γράψουμε (num 50).

(deftemplate)

- Ορισμός προτύπου για γεγονότα
- `(deftemplate <template_name> [<comment>]
 <slot-definition>)`

Πχ. `(deftemplate person
 (slot name)
 (slot age)
 (slot height))`

(deffacts)

- Ορισμός συνόλου γεγονότων
- **(deffacts <facts_name> [<comment>]
 <facts>)**

Πχ. (deffacts various_facts
 (person (name John) (age 18) (height 175))
 (person (name Ann) (age 25) (height 165))
 (person (name Paul) (age 32) (height 180))
 (birthdate (year 2020) (month 4) (day 20))
 (birthdate (year 2019) (month 6) (day 17)))

Τα γεγονότα δεν είναι υποχρεωτικό να είναι ίδιου τύπου(template).

Συνήθως τα deffacts ορίζονται σε αρχείο (.clp) και φορτώνονται από το αρχείο στο περιβάλλον CLIPS με την εντολή (load file.clp).

Η εντολή reset μεταφέρει στην fact list τα γεγονότα των deffacts.

Προσθήκη Γεγονότων

- Τα γεγονότα τοποθετούνται στη λίστα (ή μνήμη) γεγονότων (fact-list).
- Προσθήκη γεγονότος: εντολή **assert(fact)**

```
-CLIPS> (assert(snowing))
```

```
-CLIPS> (assert(snowing Jan11))
```

- Μπορούμε να δούμε το περιεχόμενο της λίστας γεγονότων

```
-CLIPS> (facts)
```

```
-f-0      (initial-fact)
```

```
-f-1      (snowing)
```

```
-f-2      (snowing Jan11)
```

```
-For a total of 3 facts.
```

```
-CLIPS>
```

- Κάθε γεγονός έχει μια διεύθυνση στη fact list (π.χ. f-2)
- Κάθε φορά που προστίθεται ένα γεγονός (με assert) θεωρούμε ότι τοποθετείται στην **επόμενη κενή θέση** της fact-list (όχι σε ενδιάμεσες κενές θέσεις που τυχόν έχουν δημιουργηθεί)

Διαγραφή Γεγονότων

- Εντολή retract
 - **(retract <fact-address>)**
 - CLIPS> (retract 1)
 - CLIPS> (facts)
 - f-0 (initial-fact)
 - f-2 (snowing Jan11)
 - For a total of 2 facts.
 - CLIPS>
- Μπορεί να έχει ως ορίσματα τις διευθύνσεις πολλών γεγονότων
 - CLIPS> (retract 0 2)
 - CLIPS> (facts)
 - CLIPS>

Σχόλια στην CLIPS

- Ξεκινούν με “;”. Οποιοδήποτε κείμενο έπεται στην ίδια γραμμή παραλείπεται
; This is an inline comment example
- Επιτρέπονται και επεξηγηματικά σχόλια που εμφανίζονται στους ορισμούς γεγονότων (deftemplate) και κανόνων (defrule)

```
(defrule my-rule "my comment"  
  (initial-fact)  
  =>  
  (printout t "Hello" crlf)  
)
```

Αριθμητικές πράξεις

- prefix format: π.χ. $(+ \ 2 \ 3)$
- Συνήθεις τελεστές: $'+', '-', '*', '/', '**'$
(ύψωση σε δύναμη)
- $(+ \ 2 \ (* \ 3 \ 4))$ δίνει 14
- $(* \ (+ \ 2 \ 3) \ 4)$ δίνει 20
- η αποτίμηση από τις εσωτερικές προς τις εξωτερικές παρενθέσεις

Κανόνες (rules)

- LHS = Left Hand Side: συνθήκες (patterns)
- RHS = Right Hand side: ενέργειες (actions)
- Ένας κανόνας γίνεται ενεργός (triggered) όταν ικανοποιούνται οι συνθήκες του από τα γεγονότα της fact-list.
- Όταν ένας ενεργός κανόνας εκτελείται (fires), στην ουσία εκτελούνται οι ενέργειές του.
- Οι ενέργειες ενός κανόνα είναι είτε φυσικές ενέργειες (π.χ. print) είτε η τροποποίηση της λίστας γεγονότων (προσθήκη, διαγραφή γεγονότων)
- Η στρατηγική επίλυσης συγκρούσεων καθορίζει ποιος ενεργός κανόνας θα εκτελεστεί.

Ορισμός κανόνων: defrule

- LHS => RHS
- Syntax:
- ```
(defrule <rule-name>
 [<comment>]
 [<declaration>] ; salience
 <patterns> ; LHS, premises, patterns,
 ; conditions, antecedent
=>
 <actions>) ; RHS, actions, consequent
```

# Παραδείγματα Κανόνων

```
(defrule snowing
 (snowing hard)
=>
 (assert(cancel class)))
```

```
(defrule car_problem_rule_1
 (car_problem (name ignition_key) (status on))
 (car_problem (name engine) (status not_start))
=>
 (assert (car_problem (name starter) (status faulty))))
```

# Initial-fact

- Γεγονός ειδικού τύπου: `initial-fact`.
- Εισάγεται αυτόματα από το σύστημα (εντολή `reset`) και μπορεί να πυροδοτήσει την εκτέλεση ενός προγράμματος σε περίπτωση που δεν υπάρχουν αρχικά γεγονότα στην `fact-list`.
- Ένας κανόνας χωρίς LHS ενεργοποιείται από `initial-fact`.

# Η Agenda

- Το σύνολο των ενεργών κανόνων σε κάθε βήμα εκτέλεσης του προγράμματος ονομάζεται agenda
- Η εντολή agenda δείχνει το περιεχόμενο της agenda τη δεδομένη στιγμή:  
—CLIPS> (agenda)



# Agenda (Παράδειγμα)

```
-CLIPS> (defrule snowing
 (snowing hard) => (assert(cancel class)))
-CLIPS> (agenda)
-CLIPS> (defrule snowing2
- (snowing hard) => (assert(alert maintenance)))
-CLIPS> (agenda)
-CLIPS> (assert(snowing hard))
-<Fact-0>
-CLIPS> (agenda)
-0 snowing: f-0
-0 snowing2: f-0
-For a total of 2 activations.
```

# “Hello World” στην CLIPS

```
(defrule start
 (initial-fact)
=>
 (printout t "Hello, world!" crlf))
```

# Δημιουργία και εκτέλεση προγράμματος

Η τυπική διαδικασία είναι:

- Γράφουμε και αποθηκεύουμε τον κώδικα σε αρχείο (π.χ. `hello-world.clp`)
- Ξεκινάμε την CLIPS και εκτελούμε τις εντολές
  - `(load όνομα-αρχείου)` π.χ. `(load hello-world.clp)`
  - `(reset)`
  - `(run)`


# Μεταβλητές (variables)

- Ονομα μεταβλητής: αποτελείται από ? και έναν ή περισσότερους χαρακτήρες
- Π.χ. ?height, ?x, ?y, ?name
- Οι μεταβλητές εμφανίζονται στο LHS των κανόνων (patterns) και λαμβάνουν τιμές κατά το ταίριασμα των patterns με τα γεγονότα στη fact-list. Στη συνέχεια οι τιμές των μεταβλητών χρησιμοποιούνται στο RHS (ενέργειες) των κανόνων.
- Μέσω των μεταβλητών επιτυγχάνεται μεταφορά πληροφορίας από το LHS στο RHS.

# Παραδείγματα


```
(defrule grandfather
 (is-a-grandfather ?name)
=>
 (assert (is-a-man ?name))
)
```

```
f1 (is-a-grandfather John)
f2 (is-a-grandfather Tom)
f3 (is-a-man John)
f4 (is-a man Tom)
```



```
(defrule grandfather
 (is-a-grandfather ?name)
=>
 (assert (is-a-father ?name))
 (assert (is-a-man ?name))
 (printout t ?name " is
grandfather" crlf)
)
```

```
f1 (is-a-grandfather John)
f2 (is-a-grandfather Tom)
f3 (is-a-father John)
f4 (is-a-man John)
f5 (is-a-father Tom)
f6 (is-a-man Tom)
```



# Παραδείγματα

```
(deftempate person (slot name) (slot hair) (slot eyes))
```

```
(deffacts person_list
```

```
(person (name John) (hair blonde) (eyes blue))
```

```
(person (name Ann) (hair black) (eyes brown))
```

```
(person (name Tom) (hair blonde) (eyes green))
```

```
(person (name Bob) (hair black) (eyes brown))
```

```
)
```

```
(defrule print_names_black_hair
```

```
(person (name ?s) (hair black) (eyes ?e))
```

```
=>
```

```
(printout t ?s)
```

```
)
```

```
(defrule print_names_black_hair_brown_eyes
```

```
(person (name ?s) (hair black) (eyes brown))
```

```
=>
```

```
(printout t ?s)
```

```
)
```

# Fact Address

- Η διεύθυνση ενός γεγονότος μπορεί στο LHS ενός κανόνα να ανατεθεί σε μια μεταβλητή (όπως οι δείκτες στη C) χρησιμοποιώντας τον συμβολισμό ('<-'). Στη συνέχεια η διεύθυνση μπορεί να χρησιμοποιηθεί στο LHS του κανόνα για την αφαίρεση του γεγονότος αυτού.
- Π.χ. `?addr <- (class (number ?cmp))`
  - Κατά το ταίριασμα με κάποιο γεγονός τύπου `class` στη `fact-list`, η μεταβλητή `?addr` παίρνει ως τιμή την διεύθυνση του γεγονότος. Στο LHS κανόνα μπορεί να γίνει αφαίρεση του γεγονότος αυτού με την εντολή `(retract ?addr)`.

# Διαγραφή γεγονότων με χρήση μεταβλητών διεύθυνσης

```
(defrule change-grandfather-fact
 ?old-fact <- (is-a-grandfather ?name)
=>
```

```
 (retract ?old-fact)
 (assert (has-a-grandchild ?name))
 (assert (is-a-man ?name))
)
```

```
f1 (is-a-grandfather John)
f2 (is-a-grandfather Tom)
f3 (has-a-grandchild John)
f4 (is-a-man John)
f5 (has-a-grandchild Tom)
f6 (is-a-man Tom)
```



# Διαγραφή Γεγονότων

- Μπορούμε να αφαιρέσουμε πολλά γεγονότα με μια εντολή retract:

```
(retract ?fact1 ?fact2 ?fact3)
```

- Μπορούμε να αφαιρέσουμε όλα τα γεγονότα:

```
(retract *)
```

# Input/Output

- Εκτύπωση στο STDOUT: (`printout t ...`)
- New line: `CrLf`
- Διάβασμα από STDIN: (`read`)
- File I/O
  - (`load <filename>`)
  - (`save <filename>`)

# Test Pattern

- Χρησιμοποιείται στο LHS για τον έλεγχο μιας συνθήκης.
- Γενική σύνταξη:

```
(test <predicate-function>)
```

- Παράδειγμα:

```
(test (> ?size 1))
```

# Μηχανισμός Εκτέλεσης Προγράμματος


- Αν οι συνθήκες (patterns) στο LHS ενός κανόνα ταιριάζουν με τα γεγονότα στη λίστα γεγονότων, τότε ο κανόνας γίνεται ενεργός και μπαίνει στην agenda.
- Σε κάθε βήμα επιλέγεται ένας κανόνας από την agenda και εκτελούνται οι ενέργειές του. Είναι δυνατή η ανάθεση (προγραμματιστικά) προτεραιοτήτων στους κανόνες (rule saliency)
- Το πρόγραμμα τερματίζει όταν δεν υπάρχουν κανόνες στην agenda.
- Είναι επιθυμητό το τελικό αποτέλεσμα της εκτέλεσης ενός προγράμματος να μην εξαρτάται από τη σειρά εκτέλεσης των κανόνων
- Προτεραιότητες βάζουμε εάν υπάρχει απόλυτη ανάγκη

# Μηχανισμός Εκτέλεσης Προγράμματος

- **Refraction (καταστολή):** κάθε κανόνας ενεργοποιείται μία μόνο φορά από τα γεγονότα σε συγκεκριμένες θέσεις στη λίστα γεγονότων (αποφυγή ατέρμονων βρόχων)
- Εάν θέλουμε ένας κανόνας να εκτελείται συνεχώς θα πρέπει τα γεγονότα που τον 'οπλίζουν' να διαγράφονται και να προστίθενται ξανά στη λίστα γεγονότων
- Π.χ.

```
(defrule simple-loop
 ?old-fact <- (loop-fact)
=>
 (printout t "Looping!" crlf)
 (retract ?oldfact)
 (assert (loop-fact)))
```

|    |                        |
|----|------------------------|
| f1 | <del>(loop-fact)</del> |
| f2 | <del>(loop-fact)</del> |
| f3 | <del>(loop-fact)</del> |
| f4 | (loop-fact)            |



# Παράδειγμα (απαρίθμηση)

Να γραφτεί κανόνας που να τυπώνει φυσικούς από 1 έως 4.

```
(deftemplate count (slot count))
```

(ορισμός γεγονότος τύπου count (αριθμός/μετρητής)

```
(deffacts initial-information
```

```
 (count 1)) (αρχικοποίηση μετρητή)
```

```
)
```

```
(defrule count_numbers
```

```
 ?adc <= (count ?c)
```

```
 (test <= ?c 4)
```

```
=>
```

```
 (printout t ?c)
```

```
 (retract ?adc)
```

```
 (assert (count + ?c 1))
```

```
)
```

f1 ~~(count 1)~~

f2 ~~(count 2)~~

f3 ~~(count 3)~~

f4 ~~(count 4)~~

f5 (count 5)

1, 2, 3, 4

# Παράδειγμα (αθροισμα εμβαδών)

```
(deftemplate rect (slot h) (slot w))
```

(ορισμός γεγονότος τύπου rect (rectangle) με τιμές h (height), w (width))

```
(deftemplate sum (slot sum))
```

(καταχώρηση του αθροίσματος)

```
(def facts initial-information
```

```
 (rect (h 10) (w 6))
```

```
 (rect (h 7) (w 5))
```

```
 (rect (h 6) (w 8))
```

```
 (rect (h 2) (w 5))
```

```
 (sum 0)) (αρχικοποίηση του αθροίσματος)
```

(γεγονότα τύπου rect προς άθροιση)

## Παράδειγμα (αθροισμα εμβαδών)

```
(defrule sum_rectangles
 (rect (h ?ht) (w ?wt))
 ?ads<-(sum ?s)
=>
 (retract ?ads)
 (assert (sum (+ (?s * (?ht ?wt))))))
)
```

f1 (rect (h 10) (w 6))  
f2 (rect (h 7) (w 5))  
f3 (rect (h 6) (w 8))  
f4 (rect (h 2) (w 5))  
~~f5 (sum 0)~~  
~~f6 (sum 60)~~  
f7 (sum 120)

Αυτός ο κανόνας είναι λάθος!

προσθέτει συνέχεια στο sum το εμβαδό  
του πρώτου γεγονότος rect



# Παράδειγμα (αθροισμα εμβαδών)

Για να λειτουργεί σωστά πρέπει να αφαιρείται το γεγονός rectangle που προστίθεται κάθε φορά.

```
(defrule sum_rectangles
 ?adr<-(rect (h ?ht) (w ?wt))
 ?ads<-(sum ?s)
```

```
=>
 (retract ?ads ?adr)
 (assert (sum (+ ?s (* ?ht ?wt))))
)
```

Αν δεν θέλουμε να αφαιρούνται τα γεγονότα rectangle;

```
f1 (rect (h 10) (w 6))
f2 (rect (h 7) (w 5))
f3 (rect (h 6) (w 8))
f4 (rect (h 2) (w 5))
f5 (sum 0)
f6 (sum 60)
f7 (sum 95)
f8 (sum 143)
f9 (sum 153)
```

# Παράδειγμα (αθροισμα εμβαδών)

Δημιουργούμε **προσωρινά γεγονότα-αντίγραφα** των γεγονότων τύπου `rect` και εφαρμόζουμε τον κανόνα σε αυτά. Τα προσωρινά γεγονότα διαγράφονται καθώς χρησιμοποιούνται.

```
(deftemplate tmp_rect (slot h) (slot w))
R1:(defrule copy_rect
 (rect (h ?ht) (w ?wt))

=>
 (assert (tmp_rect (h ?ht) (w ?wt))))
)
R2:(defrule sum_tmp_rect
 ?adr<-(tmp_rect (h ?ht) (w ?wt))
 ?ads<-(sum ?s)

=>
 (retract ?ads ?adr)
 (assert (sum (+ ?s (* ?ht ?wt))))
)
```

# Παράδειγμα (αθροισμα εμβαδών)

**R1**

```
(defrule copy_rect
 (rect (h ?ht) (w ?wt))
=>
 (assert (tmp_rect (h ?ht) (w ?wt))))
)
```

**R2**

```
(defrule sum_tmp_rect
 ?adr<-(tmp_rect (h ?ht) (w ?wt))
 ?ads<-(sum ?s)
=>
 (retract ?ads ?adr)
 (assert (sum (+ ?s (* ?ht ?wt))))
)
```

f1 (rect (h 10) (w 6))

f2 (rect (h 7) (w 5))

f3 (rect (h 6) (w 8))

f4 (rect (h 2) (w 5))

f5 (~~sum 0~~)

f6 (~~tmp\_rect (h 10) (w 6)~~)

f7 (~~tmp\_rect (h 7) (w 5)~~)

f8 (~~tmp\_rect (h 6) (w 8)~~)

f9 (~~tmp\_rect (h 2) (w 5)~~)

f10 (~~sum 60~~)

f11 (~~sum 95~~)

f12 (~~sum 143~~)

f13 (sum 153)

Το τελικό άθροισμα είναι ανεξάρτητο της  
σειράς που θα εκτελεστούν οι δύο  
κανόνες.

# Παραδείγματα

```
(deftempate person (slot name) (slot hair) (slot eyes))
```

```
(def facts person_list
```

```
(person (name John) (hair blonde) (eyes blue))
```

```
(person (name Ann) (hair black) (eyes brown))
```

```
(person (name Tom) (hair blonde) (eyes green))
```

```
(person (name Bob) (hair black) (eyes brown)))
```

```
(def facts counter_initialisation)
```

```
(count 0))
```

```
(defrule count_blonde_persons
```

```
 ?adp<-(person (name ?s) (hair blonde) (eyes ?e))
```

```
 ?adc<-(count ?c)
```

```
=>
```

```
 (retract ?adp ?adc)
```

```
 (assert(count + ?c 1)
```

```
)
```

Ο κανόνας διαγράφει τα γεγονότα person