Clips: Restricciones y funciones

José A. Alonso y Francisco J. Martín

Ciencias de la Computación e Inteligencia Artificial

Universidad de Sevilla

Restricción negativa

- Sesión

```
CLIPS> (defrule pasar
         (luz verde)
         =>
         (printout t "Se puede pasar." crlf))
CLIPS> (defrule no-pasar
         (luz ~verde)
         =>
         (printout t "No se debe pasar." crlf))
CLIPS> (assert (luz roja))
<Fact-0>
CLIPS> (run)
No se debe pasar.
CLIPS> (assert (luz verde))
<Fact-1>
CLIPS> (run)
Se puede pasar.
CLIPS>
```

Restricción disjuntiva

- Sesión

```
CLIPS> (clear)
CLIPS> (defrule precaucion
         (luz ambar|ambar-parpadeante)
         =>
         (printout t "Cruzar con precaucion." crlf))
CLIPS> (assert (luz ambar))
<Fact-0>
CLIPS> (run)
Cruzar con precaucion.
CLIPS> (assert (luz roja))
<Fact-1>
CLIPS> (run)
CLIPS> (assert (luz ambar-parpadeante))
<Fact-2>
CLIPS> (run)
Cruzar con precaucion.
CLIPS>
```

Restricciones conjuntivas

- Sesión

```
CLIPS> (clear)
CLIPS> (defrule precaucion
         (luz ?color&ambar|ambar-parpadeante)
         =>
         (printout t "Cruzar con precaucion" crlf)
         (printout t "La luz es " ?color crlf))
CLIPS> (defrule no-es-ambar-ni-roja
         (luz ?color&~ambar&~roja)
         =>
         (printout t "Cruzar" crlf)
         (printout t "La luz es " ?color crlf))
CLIPS> (assert (luz roja))
<Fact-0>
CLIPS> (run)
CLIPS> (assert (luz ambar))
<Fact-1>
CLIPS> (run)
Cruzar con precaucion
La luz es ambar
CLIPS> (assert (luz verde))
<Fact-2>
CLIPS> (run)
Cruzar
La luz es verde
CLIPS>
```

Combinación de restricciones (I)

- Fichero `ej-1.clp`

```
(deftemplate persona
  (slot nombre)
  (slot ojos)
  (slot pelo))

(deffacts personas
  (persona (nombre Ana)    (ojos verdes) (pelo rubio))
  (persona (nombre Juan)   (ojos negros) (pelo rojo))
  (persona (nombre Luis)   (ojos negros) (pelo rubio))
  (persona (nombre Blanca) (ojos azules) (pelo blanco)))

(defrule busca-personas
  (persona (nombre ?nombre1)
           (ojos ?ojos1&azules|verdes))
  (persona (nombre ?nombre2&~?nombre1)
           (ojos negros))
  =>
  (printout t ?nombre1
              " tiene los ojos " ?ojos1 crlf)
  (printout t ?nombre2
              " tiene los ojos negros" crlf)
  (printout t "-----------------------" crlf))
```

Combinación de restricciones (II)

```
CLIPS> (clear)
CLIPS> (unwatch compilations)
CLIPS> (load "ej-1.clp")
%$*
TRUE
CLIPS> (reset)
CLIPS> (facts)
f-0     (initial-fact)
f-1     (persona (nombre Ana) (ojos verdes) (pelo rubio))
f-2     (persona (nombre Juan) (ojos negros) (pelo rojo))
f-3     (persona (nombre Luis) (ojos negros) (pelo rubio))
f-4     (persona (nombre Blanca) (ojos azules) (pelo blanco))
For a total of 5 facts.
CLIPS> (agenda)
0      busca-personas: f-4,f-3
0      busca-personas: f-4,f-2
0      busca-personas: f-1,f-3
0      busca-personas: f-1,f-2
For a total of 4 activations.
CLIPS> (run)
Blanca tiene los ojos azules
Luis tiene los ojos negros
-----------------------
Blanca tiene los ojos azules
Juan tiene los ojos negros
-----------------------
Ana tiene los ojos verdes
Luis tiene los ojos negros
-----------------------
Ana tiene los ojos verdes
Juan tiene los ojos negros
-----------------------
```

Aritmética elemental (I)

- Ejemplo 1

```
CLIPS> (defrule suma
         (numeros ?x ?y ?z)
         =>
         (assert (respuesta-suma (+ ?x ?y ?z))))
CLIPS> (assert (numeros 2 3 4))
<Fact-0>
CLIPS> (run)
CLIPS> (facts)
f-0     (numeros 2 3 4)
f-1     (respuesta-suma 9)
For a total of 2 facts.
CLIPS>
```

- Ejemplo 2

```
CLISP> (clear)
CLIPS> (defrule suma
         (numeros ?x ?y)
         =>
         (printout t "La suma de " ?x
                     " y " ?y
                     " es " (+ ?x ?y) crlf))
CLIPS> (assert (numeros 2 3))
<Fact-0>
CLIPS> (run)
La suma de 2 y 3 es 5
CLISP>
```

Aritmética elemental (II)

- Fichero `ej-3.clp`

```
(deftemplate triangulo
  (slot nombre)
  (multislot lados))

(deffacts triangulos
  (triangulo (nombre A) (lados 3 4))
  (triangulo (nombre B) (lados 6 8)))

(defrule hipotenusa
  (triangulo (nombre ?n) (lados ?x ?y))
  =>
  (printout t "La hipotenusa del triangulo " ?n
      " mide " (sqrt (+ (** ?x 2) (** ?y 2))) crlf))
```

- Sesión

```
CLIPS> (clear)
CLIPS> (load "ej-3.clp")
%$*
TRUE
CLIPS> (reset)
CLIPS> (run)
La hipotenusa del triangulo B mide 10.0
La hipotenusa del triangulo A mide 5.0
```

Asignación simple

- Sesión

```
CLIPS> (defrule suma
         (numeros ?x ?y ?z)
         =>
         (bind ?respuesta-suma (+ ?x ?y ?z))
         (printout t "El resultado de la suma es: "
                     ?respuesta-suma crlf))
CLIPS> (assert (numeros 2 3 5))
<Fact-0>
CLIPS> (run)
El resultado de la suma es: 10
CLIPS>
```

- Comentarios
  - (bind <variable> <expresion>)

Asignación múltiple

- Sesión

```
CLIPS> (clear)
CLIPS> (defrule multiples-valores
         =>
         (bind ?lista1 (create$ Uno Dos Tres))
         (bind ?lista2 (create$))
         (printout t
             " La primera lista es: " ?lista1 crlf
             " La segunda lista es: " ?lista2 crlf))
CLIPS> (reset)
CLIPS> (agenda)
0      multiples-valores: f-0
For a total of 1 activation.
CLIPS> (run)
La primera lista es: (Uno Dos Tres)
La segunda lista es: ()
CLIPS>
```

- Comentarios
  - (bind <variable> <expresion>*)
  - (create$ <expresion>*)

Suma de valores (I)

- Fichero `ej-4.clp`

```
(deftemplate rectangulo
  (slot base)
  (slot altura))

(deffacts informacion-inicial
  (rectangulo (base 9) (altura 6))
  (rectangulo (base 7) (altura 5))
  (rectangulo (base 6) (altura 8))
  (rectangulo (base 2) (altura 5))
  (suma 0))

(defrule suma-areas-de-rectangulos
  (rectangulo (base ?base) (altura ?altura))
  ?suma <- (suma ?total)
  =>
  (retract ?suma)
  (assert (suma (+ ?total (* ?base ?altura)))))
```

Suma de valores (II)

- **Sesión**

```
CLIPS> (clear)
CLIPS> (load "ej-4.clp")
%$*
TRUE
CLIPS> (watch facts)
CLIPS> (reset)
==> f-0     (initial-fact)
==> f-1     (rectangulo (base 9) (altura 6))
==> f-2     (rectangulo (base 7) (altura 5))
==> f-3     (rectangulo (base 6) (altura 8))
==> f-4     (rectangulo (base 2) (altura 5))
==> f-5     (suma 0)
CLIPS> (run)
<== f-5     (suma 0)
==> f-6     (suma 54)
<== f-6     (suma 54)
==> f-7     (suma 108)
<== f-7     (suma 108)
==> f-8     (suma 162)
<== f-8     (suma 162)
...
```

Suma de valores (III)

- Fichero `ej-5.clp`

```
(deftemplate rectangulo
  (slot base)
  (slot altura))

(deffacts informacion-inicial
  (rectangulo (base 9) (altura 6))
  (rectangulo (base 7) (altura 5))
  (rectangulo (base 6) (altura 8))
  (rectangulo (base 2) (altura 5))
  (suma 0))

(defrule areas
  (rectangulo (base ?b) (altura ?h))
  =>
  (assert (area-a-sumar (* ?b ?h))))

(defrule suma-areas-de-rectangulos
  ?nueva-area <- (area-a-sumar ?area)
  ?suma <- (suma ?total)
  =>
  (retract ?suma ?nueva-area)
  (assert (suma (+ ?total ?area))))
```

Suma de valores (IV)

- **Sesión**

```
CLIPS> (clear)
CLIPS> (load "ej-5.clp")
%$**
TRUE
CLIPS> (watch facts)
CLIPS> (reset)
==> f-0     (initial-fact)
==> f-1     (rectangulo (base 9) (altura 6))
==> f-2     (rectangulo (base 7) (altura 5))
==> f-3     (rectangulo (base 6) (altura 8))
==> f-4     (rectangulo (base 2) (altura 5))
==> f-5     (suma 0)
CLIPS> (run)
==> f-6     (area-a-sumar 10)
<== f-5     (suma 0)
<== f-6     (area-a-sumar 10)
==> f-7     (suma 10)
==> f-8     (area-a-sumar 48)
<== f-7     (suma 10)
<== f-8     (area-a-sumar 48)
==> f-9     (suma 58)
==> f-10    (area-a-sumar 35)
<== f-9     (suma 58)
<== f-10    (area-a-sumar 35)
==> f-11    (suma 93)
==> f-12    (area-a-sumar 54)
<== f-11    (suma 93)
<== f-12    (area-a-sumar 54)
==> f-13    (suma 147)
```

Definición de funciones (I)

- Ejemplo 1

```
CLIPS> (deffunction hipotenusa
         (?a ?b)
         (sqrt (+ (* ?a ?a) (* ?b ?b))))
CLIPS> (defrule calcula-hipotenusa
         (triangulo ?x ?y)
         =>
         (printout t "Hipotenusa = "
                     (hipotenusa ?x ?y) crlf))
CLIPS> (assert (triangulo 3 4))
<Fact-0>
CLIPS> (run)
Hipotenusa = 5.0
CLIPS>
```

- Ejemplo 2

```
CLIPS> (deffunction contador
         ($?arg)
         (length $?arg))
CLIPS> (contador 1 2 3 color "Rojo")
5
CLIPS>
```

Definición de funciones (II)

- Ejemplo 3

```
CLISP> (deffunction factorial (?x)
         (if (= ?x 0)
             then 1
             else (* ?x (factorial (- ?x 1)))))
CLIPS> (factorial 5)
120
CLIPS> (ppdeffunction factorial)
(deffunction MAIN::factorial
   (?x)
   (if (= ?x 0)
      then
      1
      else
      (* ?x (factorial (- ?x 1)))))
CLIPS>
```

- Sintaxis

```
(deffunction <nombre> [<comentario>]
  (<parametro>*)
  <accion>*)

<parametro> ::= <variable-simple> |
                <variable-multiple>
```