Tema 9:
Clips: Restricciones y funciones

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`

```
(deffacts personas
  (persona Ana    verdes rubio)
  (persona Juan   negros rojo)
  (persona Luis   negros rubio)
  (persona Blanca azules blanco))

(defrule busca-personas
  (persona ?nombre1
           ?ojos1&azules|verdes
           ?pelo1&~negro)
  (persona ?nombre2&~?nombre1
           ?ojos2&~?ojos1
           ?pelo2&rojo|?pelo1)
  =>
  (printout t ?nombre1
              " tiene los ojos " ?ojos1
              " y el pelo " ?pelo1 crlf)
  (printout t ?nombre2
              " tiene los ojos " ?ojos2
              " y el pelo " ?pelo2 crlf)
  (printout t "-----------------------" crlf))
```

Combinación de restricciones (II)

```
CLIPS> (load "ej-1.clp")
CLIPS> Defining deffacts: personas
Defining defrule: busca-personas +j+j
TRUE
CLIPS> (reset)
CLIPS> (facts)
f-0     (initial-fact)
f-1     (persona Ana verdes rubio)
f-2     (persona Juan negros rojo)
f-3     (persona Luis negros rubio)
f-4     (persona Blanca azules blanco)
For a total of 5 facts.
CLIPS> (ppdefrule busca-personas)
(defrule MAIN::busca-personas
  (persona ?nombre1 ?ojos1&azules|verdes
           ?pelo1&~negro)
  (persona ?nombre2&~?nombre1 ?ojos2&~?ojos1
           ?pelo2&rojo|?pelo1)
  =>
  (printout t ?nombre1
              " tiene los ojos " ?ojos1
              " y el pelo " ?pelo1 crlf)
  (printout t ?nombre2
              " tiene los ojos " ?ojos2
              " y el pelo " ?pelo2 crlf)
  (printout t "-----------------------" crlf))
CLIPS> (agenda)
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 3 activations.
```

Combinación de restricciones (III)

```
CLIPS> (run)
Blanca tiene los ojos azules y el pelo blanco
Juan tiene los ojos negros y el pelo rojo
-------------------------
Ana tiene los ojos verdes y el pelo rubio
Luis tiene los ojos negros y el pelo rubio
-------------------------
Ana tiene los ojos verdes y el pelo rubio
Juan tiene los ojos negros y el pelo rojo
-------------------------
CLIPS>
```

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`

```
(deffacts triangulos
  (triangulo 1 lados 3 4)
  (triangulo 2 lados 6 8)
  (triangulo 3 lados 1 2))

(deffacts almacen
  (almacenado A 2.0)
  (almacenado B 5.0)
  (almacenado C 10.0))

(defrule hipotenusa
  (triangulo ?n lados ?x ?y)
  (almacenado ?ID =(sqrt (+ (** ?x 2) (** ?y 2))))
  =>
  (printout t "La hipotenusa del triangulo " ?n
              " esta en el registro " ?ID crlf))
```

Aritmética elemental (III)

- Sesión

```
CLISP> (clear)
CLIPS> (load "ej-3.clp")
CLIPS> Defining deffacts: triangulos
Defining deffacts: almacen
Defining defrule: hipotenusa +j+j
TRUE
CLIPS> (reset)
CLIPS> (run)
La hipotenusa del triangulo 2 esta en el registro C
La hipotenusa del triangulo 1 esta en el registro B
CLIPS>
```

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`

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

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

Suma de valores (II)

- Sesión

```
CLIPS> (clear)
CLIPS> (load "ej-4.clp")
CLIPS> Defining deffacts: informacion-inicial
Defining defrule: suma-rectangulos +j+j
TRUE
CLIPS> (watch facts)
CLIPS> (reset)
==> f-0     (initial-fact)
==> f-1     (rectangulo 9 6)
==> f-2     (rectangulo 7 5)
==> f-3     (rectangulo 6 8)
==> f-4     (rectangulo 2 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`

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

(defrule suma-rectangulos
  (rectangulo ?base ?altura)
  =>
  (assert (area-a-sumar (* ?base ?altura))))

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

Suma de valores (IV)

- Sesión

```
CLIPS> (clear)
CLIPS> (load "ej-5.clp")
CLIPS> Defining deffacts: informacion-inicial
Defining defrule: suma-rectangulos +j
Defining defrule: suma-areas +j+j
TRUE
CLIPS> (reset)
CLIPS> (watch facts)
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)
CLIPS>
```

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>
```