

# Tema 8: Elementos condicionales y acciones procedimentales

José A. Alonso Jiménez  
Miguel A. Gutiérrez Naranjo  
Francisco J. Martín Mateos

Dpto. de Ciencias de la Computación e Inteligencia Artificial

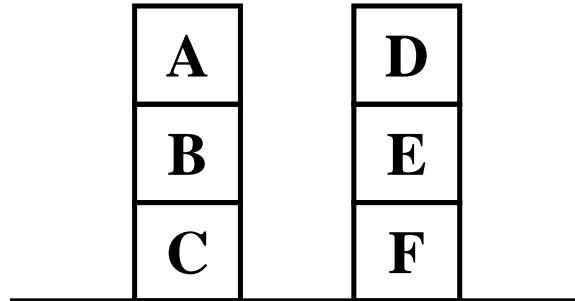
UNIVERSIDAD DE SEVILLA

# Contenido y bibliografía

- Ejemplo: Mundo de los bloques
- Elementos condicionales
  - Disyunción
  - Conjunción
  - Cuantificador existencial
  - Cuantificador universal
- Lectura de datos
- Acciones procedimentales
  - Condicional *if*
  - Bucle *while*
- Ejemplo: Nim
- Bibliografía
  - Giarratano, J.C. y Riley, G. “Expert Systems Principles and Programming (2nd ed.)” (PWS Pub. Co., 1994)
    - \* Cap. 9: “Advanced Pattern Matching”
    - \* Cap. 11.11: “Procedural Functons”
  - Giarratano, J.C. “CLIPS 6.0 User’s Guide”

# Mundo de los bloques

- Enunciado



- Objetivo: Poner C encima de E

- Representación

```
(deffacts estado-inicial
  (pila A B C)
  (pila D E F)
  (objetivo C esta-encima-del E))
```

# Mundo de los bloques

- Reglas:

```
;;; REGLA: mover-bloque-sobre-bloque
;;; SI
;;;   el objetivo es poner el bloque X encima del
;;;   bloque Y y
;;;   no hay nada encima del bloque X ni del bloque Y
;;; ENTONCES
;;;   colocamos el bloque X encima del bloque Y y
;;;   actualizamos los datos.
```

```
(defrule mover-bloque-sobre-bloque
  ?obj <- (objetivo ?blq-1 esta-encima-del ?blq-2)
  ?p-1 <- (pila ?blq-1 $?resto-1)
  ?p-2 <- (pila ?blq-2 $?resto-2)
  =>
  (retract ?ob ?p1 ?p2)
  (assert (pila $?resto-1))
  (assert (pila ?blq-1 ?blq-2 $?resto-2))
  (printout t ?blq-1 " movido encima del "
            ?blq-2 crlf))
```

# Mundo de los bloques

- Reglas:

```
;;; REGLA: mover-bloque-al-suelo
;;; SI
;;;   el objetivo es mover el bloque X al suelo y
;;;   no hay nada encima de X
;;; ENTONCES
;;;   movemos el bloque X al suelo y
;;;   actualizamos los datos.
```

```
(defrule mover-bloque-al-suelo
  ?obj <- (objetivo ?blq-1 esta-encima-del suelo)
  ?p-1 <- (pila ?blq-1 $?resto)
  =>
  (retract ?objetivo ?pila-1)
  (assert (pila ?blq-1))
  (assert (pila $?resto))
  (printout t ?blq-1 " movido encima del suelo."
            crlf))
```

# Mundo de los bloques

- Reglas:

```
;;; REGLA: libera-bloque-movible
;;; SI
;;;   el objetivo es poner el bloque X encima de Y
;;;   (bloque o suelo) y
;;;   X es un bloque y
;;;   hay un bloque encima del bloque X
;;; ENTONCES
;;;   hay que poner el bloque que está encima de X
;;;   en el suelo.
```

```
(defrule liberar-bloque-movible
  (objetivo ?bloque esta-encima-del ?)
  (pila ?cima $? ?bloque $?)
  =>
  (assert (objetivo ?cima esta-encima-del suelo)))
```

```
;;; REGLA: libera-bloque-soporte
;;; SI
;;;   el objetivo es poner el bloque X (bloque o
;;;   nada) encima de Y e
;;;   hay un bloque encima del bloque Y
;;; ENTONCES
;;;   hay que poner el bloque que está encima de Y
;;;   en el suelo.
```

```
(defrule liberar-bloque-soporte
  (objetivo ? esta-encima-del ?bloque)
  (pila ?cima $? ?bloque $?)
  =>
  (assert (objetivo ?cima esta-encima-del suelo)))
```

# Mundo de los bloques

- Sesión:

```
CLIPS> (clear)
CLIPS> (unwatch all)
CLIPS> (watch facts)
CLIPS> (watch activations)
CLIPS> (watch rules)
CLIPS> (load "bloques.clp")
$****
TRUE
CLIPS> (reset)
==> f-0      (initial-fact)
==> f-1      (pila A B C)
==> f-2      (pila D E F)
==> f-3      (objetivo C esta-encima-del E)
==> Activation 0      liberar-bloque-soporte: f-3,f-2
==> Activation 0      liberar-bloque-movible: f-3,f-1
CLIPS> (run)
FIRE      1 liberar-bloque-movible: f-3,f-1
==> f-4      (objetivo A esta-encima-del suelo)
==> Activation 0      mover-bloque-al-suelo: f-4,f-1
FIRE      2 mover-bloque-al-suelo: f-4,f-1
<== f-4      (objetivo A esta-encima-del suelo)
<== f-1      (pila A B C)
==> f-5      (pila A)
==> f-6      (pila B C)
==> Activation 0      liberar-bloque-movible: f-3,f-6
A movido encima del suelo.
```

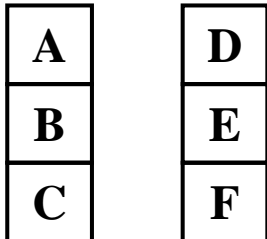
# Mundo de los bloques

## ● Sesión:

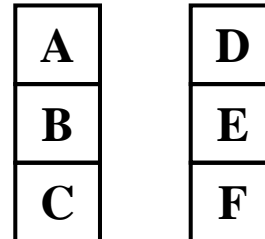
```
FIRE    3 liberar-bloque-movible: f-3,f-6
==> f-7    (objetivo B esta-encima-del suelo)
==> Activation 0    mover-bloque-al-suelo: f-7,f-6
FIRE    4 mover-bloque-al-suelo: f-7,f-6
<== f-7    (objetivo B esta-encima-del suelo)
<== f-6    (pila B C)
==> f-8    (pila B)
==> f-9    (pila C)
B movido encima del suelo.
FIRE    5 liberar-bloque-soporte: f-3,f-2
==> f-10   (objetivo D esta-encima-del suelo)
==> Activation 0    mover-bloque-al-suelo: f-10,f-2
FIRE    6 mover-bloque-al-suelo: f-10,f-2
<== f-10   (objetivo D esta-encima-del suelo)
<== f-2    (pila D E F)
==> f-11   (pila D)
==> f-12   (pila E F)
==> Activation 0
        mover-bloque-sobre-bloque: f-3,f-9,f-12
D movido encima del suelo.
FIRE    7 mover-bloque-sobre-bloque: f-3,f-9,f-12
<== f-3    (objetivo C esta-encima-del E)
<== f-9    (pila C)
<== f-12   (pila E F)
==> f-13   (pila)
==> f-14   (pila C E F)
C movido encima del E
CLIPS>
```



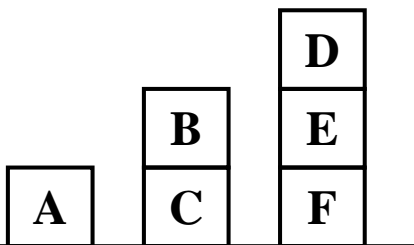
# Mundo de los bloques



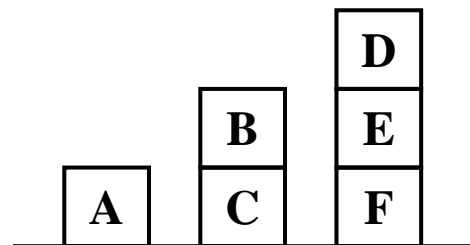
Objetivos: C/E  
Agenda: Lib C  
Lib E



Objetivos: A/Suelo  
C/E  
Agenda: Mover A  
Lib E

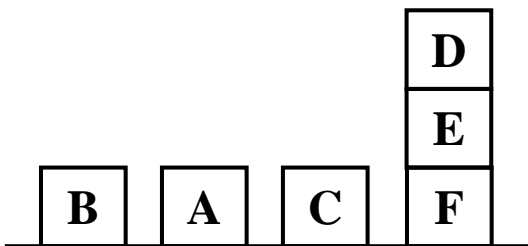


Objetivos: C/E  
Agenda: Lib C  
Lib E

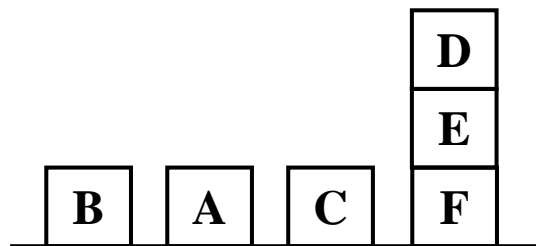


Objetivos: B/Suelo  
C/E  
Agenda: Mover B  
Lib E

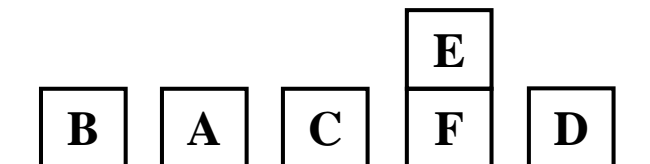
# Mundo de los bloques



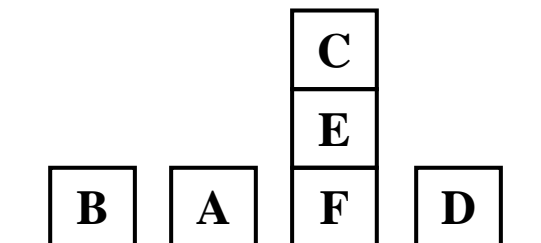
Objetivos: C/E  
Agenda: Lib E



Objetivos: D/Suelo  
C/E  
Agenda: Mover D



Objetivos: C/E  
Agenda: Mover C



# Elementos condicionales

- Reglas disyuntivas: ej-1.clp

```
(defrule no-hay-clase-1
  (festivo hoy)
  =>
  (printout t "Hoy no hay clase" crlf))
```

```
(defrule no-hay-clase-2
  (sabado hoy)
  =>
  (printout t "Hoy no hay clase" crlf))
```

```
(defrule no-hay-clase-3
  (hay-examen hoy)
  =>
  (printout t "Hoy no hay clase" crlf))
```

```
(deffacts inicio
  (sabado hoy)
  (hay-examen hoy))
```

# Elementos condicionales

- Reglas disyuntivas. Sesión:

```
CLIPS> (clear)
CLIPS> (unwatch all)
CLIPS> (watch facts)
CLIPS> (watch activations)
CLIPS> (watch rules)
CLIPS> (load "ej-1.clp")
***$
TRUE
CLIPS> (reset)
==> f-0      (initial-fact)
==> f-1      (sabado hoy)
==> Activation 0      no-hay-clase-2: f-1
==> f-2      (hay-examen hoy)
==> Activation 0      no-hay-clase-3: f-2
CLIPS> (run)
FIRE      1 no-hay-clase-3: f-2
Hoy no hay clase
FIRE      2 no-hay-clase-2: f-1
Hoy no hay clase
```

# Disyunción

- Elementos condicionales disyuntivos: ej-2.cl

```
(defrule no-hay-clase
  (or (festivo hoy)
       (sabado hoy)
       (hay-examen hoy))
  =>
  (printout t "Hoy no hay clase" crlf))

(deffacts inicio
  (sabado hoy)
  (hay-examen hoy))
```

# Disyunción

- Sesión

```
CLIPS> (clear)
CLIPS> (unwatch all)
CLIPS> (watch facts)
CLIPS> (watch activations)
CLIPS> (watch rules)
CLIPS> (load "ej-2.clp")
*$
TRUE
CLIPS> (reset)
==> f-0      (initial-fact)
==> f-1      (sabado hoy)
==> Activation 0      no-hay-clase: f-1
==> f-2      (hay-examen hoy)
==> Activation 0      no-hay-clase: f-2
CLIPS> (run)
FIRE      1 no-hay-clase: f-2
Hoy no hay clase
FIRE      2 no-hay-clase: f-1
Hoy no hay clase
CLIPS>
```

# Limitación de disparos disyuntivos

- Ejemplo: ej-3.clp

```
(defrule no-hay-clase
  ?periodo <- (periodo lectivo)
  (or (festivo hoy)
      (sabado hoy)
      (hay-examen hoy))
  =>
  (retract ?periodo)
  (assert (periodo lectivo-sin-clase))
  (printout t "Hoy no hay clase" crlf))

(deffacts inicio
  (sabado hoy)
  (hay-examen hoy))
```

# Limitación de disparos disyuntivos

## ● Sesión

```
CLIPS> (clear)
CLIPS> (unwatch all)
CLIPS> (watch facts)
CLIPS> (watch activations)
CLIPS> (watch rules)
CLIPS> (load "ej-3.clp")
*$
TRUE
CLIPS> (reset)
==> f-0      (initial-fact)
==> f-1      (sabado hoy)
==> f-2      (hay-examen hoy)
CLIPS> (assert (periodo lectivo))
==> f-3      (periodo lectivo)
==> Activation 0      no-hay-clase: f-3,f-2
==> Activation 0      no-hay-clase: f-3,f-1
CLIPS> (run)
FIRE      1 no-hay-clase: f-3,f-1
<== f-3      (periodo lectivo)
<== Activation 0      no-hay-clase: f-3,f-2
==> f-4      (periodo lectivo-sin-clase)
Hoy no hay clase
CLIPS>
```



## Limitación de disparos disyuntivos

- Programa equivalente sin elementos condicionales disyuntivos: ej-4.clp

```
(defrule no-hay-clase-1
  ?periodo <- (periodo lectivo)
  (festivo hoy)
  =>
  (retract ?periodo)
  (assert (periodo lectivo-sin-clase))
  (printout t "Hoy no hay clase" crlf))
```

```
(defrule no-hay-clase-2
  ?periodo <- (periodo lectivo)
  (sabado hoy)
  =>
  (retract ?periodo)
  (assert (periodo lectivo-sin-clase))
  (printout t "Hoy no hay clase" crlf))
```

```
(defrule no-hay-clase-3
  ?periodo <- (periodo lectivo)
  (hay-examen hoy)
  =>
  (retract ?periodo)
  (assert (periodo lectivo-sin-clase))
  (printout t "Hoy no hay clase" crlf))
```

```
(deffacts inicio
  (sabado hoy)
  (hay-examen hoy))
```

# Eliminación de causas disyuntivas

- **Ejemplo:** ej-5.clp

```
(defrule no-hay-clase
  ?periodo <- (periodo lectivo)
  (or ?causa <- (festivo hoy)
      ?causa <- (sabado hoy)
      ?causa <- (hay-examen hoy))
  =>
  (retract ?periodo ?causa)
  (assert (periodo lectivo-sin-clase))
  (printout t "Hoy no hay clase" crlf))

(deffacts inicio
  (sabado hoy)
  (hay-examen hoy)
  (periodo lectivo))
```

# Eliminación de causas disyuntivas

## ● Sesión

```
CLIPS> (clear)
CLIPS> (unwatch all)
CLIPS> (watch facts)
CLIPS> (watch activations)
CLIPS> (load "ej-5.clp")
*$
TRUE
CLIPS> (reset)
==> f-0      (initial-fact)
==> f-1      (sabado hoy)
==> f-2      (hay-examen hoy)
==> f-3      (periodo lectivo)
==> Activation 0      no-hay-clase: f-3,f-2
==> Activation 0      no-hay-clase: f-3,f-1
CLIPS> (run)
<== f-3      (periodo lectivo)
<== Activation 0      no-hay-clase: f-3,f-2
<== f-1      (sabado hoy)
==> f-4      (periodo lectivo-sin-clase)
Hoy no hay clase
CLIPS> (facts)
f-0      (initial-fact)
f-2      (hay-examen hoy)
f-4      (periodo lectivo-sin-clase)
For a total of 3 facts.
CLIPS>
```

# Conjunción

- **Conjunciones implícitas**

```
(defrule no-hay-clase-por-puente
  (festivo ayer)
  (festivo mañana)
  =>
  (printout t "Hoy no hay clase" crlf))
```

- **Conjunciones explícitas. Elementos condicionales conjuntivos:**

```
(defrule no-hay-clase-por-puente
  (and (festivo ayer)
        (festivo mañana))
  =>
  (printout t "Hoy no hay clase" crlf))
```

# Conjunción

- **Conjunciones y disyunciones: ej-6.clp**

```
(defrule no-hay-clase
  ?periodo <- (periodo lectivo)
  (or (festivo hoy)
      (sabado hoy)
      (and (festivo ayer)
           (festivo manana))))
=>
(retract ?periodo)
(assert (periodo lectivo-sin-clase))
(printout t "Hoy no hay clase" crlf))

(deffacts inicio
  (periodo lectivo)
  (festivo ayer)
  (festivo manana))
```

# Conjunción

- Sesión

```
CLIPS> (clear)
CLIPS> (unwatch all)
CLIPS> (watch facts)
CLIPS> (watch activations)
CLIPS> (load "ej-6.clp")
*$
TRUE
CLIPS> (reset)
==> f-0      (initial-fact)
==> f-1      (periodo lectivo)
==> f-2      (festivo ayer)
==> f-3      (festivo mañana)
==> Activation 0      no-hay-clase: f-1,f-2,f-3
CLIPS> (run)
<== f-1      (periodo lectivo)
==> f-4      (periodo lectivo-sin-clase)
Hoy no hay clase
CLIPS>
```

# Conjunción

- Reglas equivalentes

```
(defrule no-hay-clase-1
  ?periodo <- (periodo lectivo)
  (festivo hoy)
  =>
  (retract ?periodo)
  (assert (periodo lectivo-sin-clase))
  (printout t "Hoy no hay clase" crlf))
```

```
(defrule no-hay-clase-2
  ?periodo <- (periodo lectivo)
  (sabado hoy)
  =>
  (retract ?periodo)
  (assert (periodo lectivo-sin-clase))
  (printout t "Hoy no hay clase" crlf))
```

```
(defrule no-hay-clase-3
  ?periodo <- (periodo lectivo)
  (festivo ayer)
  (festivo manana)
  =>
  (retract ?periodo)
  (assert (periodo lectivo-sin-clase))
  (printout t "Hoy no hay clase" crlf))
```

# Cuantificación existencial

- **Elementos condicionales existenciales:**

ej-7.clp

```
(defrule no-hay-clase
  (exists (hoy ?x&es-sabado|es-festivo|hay-examen))
  =>
  (printout t "Hoy no hay clase" crlf))
```

```
(defacts inicio
  (hoy es-sabado)
  (hoy hay-examen))
```

- **Programa equivalente a ej-2.clp con limitación de disparos disyuntivos.**



# Cuantificación existencial

## ● Sesión:

```
CLIPS> (unwatch all)
CLIPS> (watch facts)
CLIPS> (watch activations)
CLIPS> (watch rules)
CLIPS> (load "ej-7.clp")
*$
TRUE
CLIPS> (reset)
==> f-0      (initial-fact)
==> Activation 0      no-hay-clase: f-0,
<== Activation 0      no-hay-clase: f-0,
==> f-1      (hoy es-sabado)
==> Activation 0      no-hay-clase: f-0,
==> f-2      (hoy hay-examen)
CLIPS> (run)
FIRE      1 no-hay-clase: f-0,
Hoy no hay clase
CLIPS> (assert (hoy es-festivo))
==> f-3      (hoy es-festivo)
<Fact-3>
CLIPS> (retract 1 2 3)
<== f-1      (hoy es-sabado)
<== f-2      (hoy hay-examen)
<== f-3      (hoy es-festivo)
CLIPS> (assert (hoy es-domingo))
==> f-4      (hoy es-domingo)
<Fact-4>
CLIPS> (assert (hoy es-festivo))
==> f-5      (hoy es-festivo)
==> Activation 0      no-hay-clase: f-0,
<Fact-5>
```

# Lectura de datos y variables globales

- **Lectura de datos:**

(read)

(readline)

- **Variables globales:**

```
(defglobal  
  ?*<simbolo>* = <valor>)
```

- **Ejemplo: Adivina el número**

```
CLIPS> (reset)  
CLIPS> (run)  
Escribe un numero: 3  
3 es bajo  
Escribe un numero: 9  
9 es alto  
Escribe un numero: 7  
7 es correcto  
CLIPS>
```

- **Código: Adivina el número**

```
(defglobal  
  ?*numero* = 7)  
  
(defrule juego  
  =>  
  (assert (lee)))
```

# Lectura de datos y variables globales

- Ejemplo: Adivina el número (I)

```
(defrule lee
  ?h <- (lee)
  =>
  (retract ?h)
  (printout t "Escribe un numero: ")
  (assert (numero (read))))
```

```
(defrule bajo
  ?h <- (numero ?n&:(< ?n ?*numero*))
  =>
  (retract ?h)
  (printout t ?n " es bajo" crlf)
  (assert (lee)))
```

```
(defrule alto
  ?h <- (numero ?n&:(> ?n ?*numero*))
  =>
  (retract ?h)
  (printout t ?n " es alto" crlf)
  (assert (lee)))
```

```
(defrule exacto
  ?h <- (numero ?n&:(= ?n ?*numero*))
  =>
  (retract ?h)
  (printout t ?n " es correcto" crlf))
```

# Lectura de hechos como cadenas

- **Sesión**

```
CLIPS> (defrule inserta-hecho
=>
  (printout t "Escribe un hecho como cadena"
            crlf)
  (assert-string (read)))
CLIPS> (reset)
CLIPS> (run)
Escribe un hecho como cadena
"(color verde)"
CLIPS> (facts)
f-0      (initial-fact)
f-1      (color verde)
For a total of 2 facts.
CLIPS>
```

- **Añadir un hecho expresado como una cadena:**

- (assert-string <cadena>)

# Lectura de líneas

- **Sesión**

```
CLIPS> (defrule lee-linea
=>
  (printout t "Introduce datos." crlf)
  (bind ?cadena (readline))
  (assert-string (str-cat "(" ?cadena ")")))
CLIPS> (reset)
CLIPS> (run)
Introduce datos.
colores verde azul ambar rojo
CLIPS> (facts)
f-0      (initial-fact)
f-1      (colores verde azul ambar rojo)
For a total of 2 facts.
CLIPS>
```

- **Concatenación:**

- (str-cat <cadena>\*)

# Acciones procedimentales: if y bind

- **Condicional:**

```
(if <condicion>
  then <accion>*
  [else <accion>*])
```

- **Asignacion:**

```
(bind <variable> <valor>)
```

- **Ejemplo: Adivina el número (II)**

```
(defrule lee
  ?h <- (lee)
  =>
  (retract ?h)
  (printout t "Escribe un numero: ")
  (bind ?n (read))
  (if (not (numberp ?n))
    then
      (printout t "Eso no es un numero." crlf)
      (assert (lee))
    else
      (assert (numero ?n))))
```

# Acciones procedimentales: while

- **Bucle:**

```
(while <condicion> do
  <accion>*)
```

- **Ejemplo: Adivina el número (III)**

```
(defrule lee
  ?h <- (lee)
  =>
  (retract ?h)
  (printout t "Escribe un numero: ")
  (bind ?n (read))
  (while (not (numberp ?n)) do
    (printout t "Eso no es un numero." crlf)
    (printout t "Escribe un numero: ")
    (bind ?n (read)))
  (assert (numero ?n)))
```

# Nim

- Sesión:

```
CLIPS> (clear)
CLIPS> (unwatch all)
CLIPS> (watch facts)
CLIPS> (watch activations)
CLIPS> (watch rules)
CLIPS> (load "nim-1.clp")
$*****$*
TRUE
CLIPS> (reset)
<== f-0      (initial-fact)
==> f-0      (initial-fact)
==> f-1      (turno h)
==> f-2      (numero-de-piezas 11)
==> Activation 0      eleccion-humana: f-1,f-2
==> f-3      (computadora-coge 1 cuando-el-resto-es 1)
==> f-4      (computadora-coge 1 cuando-el-resto-es 2)
==> f-5      (computadora-coge 2 cuando-el-resto-es 3)
==> f-6      (computadora-coge 3 cuando-el-resto-es 0)
CLIPS> (run)
FIRE      1 eleccion-humana: f-1,f-2
<== f-1      (turno h)
Quedan 11 pieza(s)
Cuantas piezas coges: 2
```



# Nim

## ● Sesión:

```
==> f-7      (eleccion-humana 2)
==> Activation 0  correcta-eleccion-humana: f-7,f-2
FIRE      2 correcta-eleccion-humana: f-7,f-2
<== f-7      (eleccion-humana 2)
<== f-2      (numero-de-piezas 11)
==> f-8      (numero-de-piezas 9)
==> f-9      (turno c)
==> Activation 0  eleccion-computadora: f-9,f-8,f-3
FIRE      3 eleccion-computadora: f-9,f-8,f-3
<== f-9      (turno c)
<== f-8      (numero-de-piezas 9)
Quedan 9 pieza(s)
La computadora coge 1 pieza(s)
==> f-10     (numero-de-piezas 8)
==> f-11     (turno h)
==> Activation 0      eleccion-humana: f-11,f-10
FIRE      4 eleccion-humana: f-11,f-10
<== f-11     (turno h)
Quedan 8 pieza(s)
Cuantas piezas coges: 3
```

# Nim

## ● Sesión:

```
==> f-12      (eleccion-humana 3)
==> Activation 0  correcta-eleccion-humana: f-12,f-10
FIRE    5 correcta-eleccion-humana: f-12,f-10
<== f-12      (eleccion-humana 3)
<== f-10      (numero-de-piezas 8)
==> f-13      (numero-de-piezas 5)
==> f-14      (turno c)
==> Activation 0  eleccion-computadora: f-14,f-13,f-3
FIRE    6 eleccion-computadora: f-14,f-13,f-3
<== f-14      (turno c)
<== f-13      (numero-de-piezas 5)
Quedan 5 pieza(s)
La computadora coge 1 pieza(s)
==> f-15      (numero-de-piezas 4)
==> f-16      (turno h)
==> Activation 0      eleccion-humana: f-16,f-15
FIRE    7 eleccion-humana: f-16,f-15
<== f-16      (turno h)
Quedan 4 pieza(s)
Cuantas piezas coges: 3
```

# Nim

- **Sesión:**

```
==> f-17      (eleccion-humana 3)
==> Activation 0  correcta-eleccion-humana: f-17,f-15
FIRE      8 correcta-eleccion-humana: f-17,f-15
<== f-17      (eleccion-humana 3)
<== f-15      (numero-de-piezas 4)
==> f-18      (numero-de-piezas 1)
==> f-19      (turno c)
==> Activation 0      pierde-la-computadora: f-19,f-18
FIRE      9 pierde-la-computadora: f-19,f-18
Queda 1 pieza
La computadora coge la ultima pieza
He perdido
CLIPS>
```

# Nim

- Código:

```
(deffacts datos-iniciales
  (turno h)
  (numero-de-piezas 11))

(defrule pierde-la-computadora
  (turno c)
  (numero-de-piezas 1)
  =>
  (printout t "Queda 1 pieza" crlf)
  (printout t "La computadora coge la ultima pieza"
            crlf)
  (printout t "He perdido" crlf))

(deffacts heuristica
  (computadora-coge 1 cuando-el-resto-es 1)
  (computadora-coge 1 cuando-el-resto-es 2)
  (computadora-coge 2 cuando-el-resto-es 3)
  (computadora-coge 3 cuando-el-resto-es 0))

(defrule eleccion-computadora
  ?turno <- (turno c)
  ?pila <- (numero-de-piezas ?n&:(> ?n 1))
  (computadora-coge ?m cuando-el-resto-es
                    =(mod ?n 4))
  =>
  (retract ?turno ?pila)
  (printout t "Quedan " ?n " pieza(s)" crlf)
  (printout t "La computadora coge " ?m
            " pieza(s)" crlf)
  (assert (numero-de-piezas (- ?n ?m))
          (turno h)))
```

# Nim

- Código:

```
(defrule pierde-el-humano
  (turno h)
  (numero-de-piezas 1)
  =>
  (printout t "Queda 1 pieza" crlf)
  (printout t "Tienes que coger la ultima pieza"
            crlf)
  (printout t "Has perdido" crlf))
```

```
(defrule eleccion-humana
  ?turno <- (turno h)
  ?pila <- (numero-de-piezas ?n&:(> ?n 1))
  =>
  (retract ?turno)
  (printout t "Quedan " ?n " pieza(s)" crlf)
  (printout t "Cuantas piezas coges: ")
  (assert (eleccion-humana (read))))
```

# Nim

- Código:

```
(defrule incorrecta-eleccion-humana
  ?eleccion <- (eleccion-humana ?m)
  (numero-de-piezas ?n&:(> ?n 1))
  (test (not (and (integerp ?m)
                  (>= ?m 1)
                  (<= ?m 3)
                  (< ?m ?n))))

=>
(retract ?eleccion)
(printout t "Tiene que elegir "
          "un numero entre 1 y 3" crlf)
(assert (turno h)))

(defrule correcta-eleccion-humana
  ?eleccion <- (eleccion-humana ?m)
  ?pila <- (numero-de-piezas ?n&:(> ?n 1))
  (test (and (integerp ?m)
             (>= ?m 1)
             (<= ?m 3)
             (< ?m ?n)))

=>
(retract ?eleccion ?pila)
(assert (numero-de-piezas (- ?n ?m))
        (turno c)))
```

# Nim con if

- **Regla:** eleccion-humana

```
(defrule eleccion-humana
  ?turno <- (turno h)
  ?pila <- (numero-de-piezas ?n&:(> ?n 1))
  =>
  (retract ?turno)
  (printout t "Quedan " ?n " pieza(s)" crlf)
  (printout t "Cuantas piezas coges: ")
  (bind ?m (read))
  (if (and (integerp ?m)
           (>= ?m 1)
           (<= ?m 3)
           (< ?m ?n))
      then
        (bind ?nuevo (- ?n ?m))
        (retract ?pila)
        (assert (numero-de-piezas ?nuevo)
                (turno c))
      else
        (printout t "Tiene que elegir "
                  "un numero entre 1 y 3" crlf)
        (assert (turno h))))
```

# Nim con while

- **Regla:** eleccion-humana

```
(defrule eleccion-humana
  ?turno <- (turno h)
  ?pila <- (numero-de-piezas ?n&:(> ?n 1))
  =>
  (retract ?turno ?pila)
  (printout t "Quedan " ?n " pieza(s)" crlf)
  (printout t "Cuantas piezas coges: ")
  (bind ?m (read))
  (while (not (and (integerp ?m)
                   (>= ?m 1)
                   (<= ?m 3)
                   (< ?m ?n))) do
    (printout t "Tiene que elegir "
              "un numero entre 1 y 3" crlf)
    (printout t "Quedan " ?n " pieza(s)" crlf)
    (printout t "Cuantas piezas coges: ")
    (bind ?m (read)))
  (assert (numero-de-piezas (- ?n ?m))
  (turno c)))
```



# Nim con funciones definidas

- Regla: eleccion-humana

```
(deffunction piezas-cogidas-de (?m ?n)
  (while (not (and (integerp ?m)
                  (>= ?m 1)
                  (<= ?m 3)
                  (< ?m ?n))) do
    (printout t "Tiene que elegir "
              "un numero entre 1 y 3" crlf)
    (printout t "Quedan " ?n " pieza(s)" crlf)
    (printout t "Cuantas piezas coges: ")
    (bind ?m (read)))
  ?m)
```

```
(defrule eleccion-humana
  ?turno <- (turno h)
  ?pila <- (numero-de-piezas ?n&:(> ?n 1))
  =>
  (retract ?turno ?pila)
  (printout t "Quedan " ?n " pieza(s)" crlf)
  (printout t "Cuantas piezas coges: ")
  (bind ?m (piezas-cogidas-de (read) ?n))
  (assert (numero-de-piezas (- ?n ?m)
                            (turno c))))
```

# Nim con acciones definidas

- **Regla:** eleccion-humana

```
(deffunction coge-piezas (?n)
  (printout t "Quedan " ?n " pieza(s)" crlf)
  (printout t "Cuantas piezas coges: ")
  (bind ?m (read))
  (while (not (and (integerp ?m)
                   (>= ?m 1)
                   (<= ?m 3)
                   (< ?m ?n))) do
    (printout t "Tiene que elegir "
              "un numero entre 1 y 3" crlf)
    (printout t "Cuantas piezas coges: ")
    (bind ?m (read)))
  (assert (numero-de-piezas (- ?n ?m))
  (turno c)))
```

```
(defrule eleccion-humana
  ?turno <- (turno h)
  ?pila <- (numero-de-piezas ?n&:(> ?n 1))
  =>
  (retract ?turno ?pila)
  (coge-piezas ?n))
```