Diseño modular y control de la ejecución

José A. Alonso y Francisco J. Martín

Ciencias de la Computación e Inteligencia Artificial

Universidad de Sevilla

Técnicas de control

- Ejemplo de fases de un problema:
  - Detección.
  - Aislamiento.
  - Recuperación.
- Técnicas de control:
  - Control empotrado en las reglas.
  - Prioridades.
  - Reglas de control.
  - Módulos.

Control empotrado en las reglas

- Fases en el Nim:
  - Elección del jugador.
  - Elección del número de piezas.
  - Turno del humano.
  - Turno de la computadora.
- Hechos de control:
  - `(fase elige-jugador)`
  - `(fase elige-numero-de-piezas)`
  - `(turno h)`
  - `(turno c)`
- Inconvenientes del control empotrado en las reglas:
  - Dificultad para entenderse.
  - Dificultad para precisar la conclusión de cada fase.

Prioridades

- Ejemplo: `ej-1.clp`

```
(deffacts inicio
  (prioridad primera)
  (prioridad segunda)
  (prioridad tercera))

(defrule regla-1
  (prioridad primera)
  =>
  (printout t "Escribe primera" crlf))

(defrule regla-2
  (prioridad segunda)
  =>
  (printout t "Escribe segunda" crlf))

(defrule regla-3
  (prioridad tercera)
  =>
  (printout t "Escribe tercera" crlf))
```

Prioridades

- Sesión

```
CLISP> (load "ej-1.clp")
CLIPS> $***
TRUE
CLIPS> (reset)
CLIPS> (facts)
f-0     (initial-fact)
f-1     (prioridad primera)
f-2     (prioridad segunda)
f-3     (prioridad tercera)
For a total of 4 facts.
CLIPS> (rules)
regla-1
regla-2
regla-3
For a total of 3 defrules.
CLIPS> (agenda)
0      regla-3: f-3
0      regla-2: f-2
0      regla-1: f-1
For a total of 3 activations.
CLIPS> (run)
Escribe tercera
Escribe segunda
Escribe primera
CLIPS>
```

Prioridades

- Ejemplo: `ej-2.clp`

```
(deffacts inicio
  (prioridad primera)
  (prioridad segunda)
  (prioridad tercera))

(defrule regla-1
  (declare (salience 30))
  (prioridad primera)
  =>
  (printout t "Escribe primera" crlf))

(defrule regla-2
  (declare (salience 20))
  (prioridad segunda)
  =>
  (printout t "Escribe segunda" crlf))

(defrule regla-3
  (declare (salience 10))
  (prioridad tercera)
  =>
  (printout t "Escribe tercera" crlf))
```

Prioridades

- Sesión

```
CLIPS> (load "ej-2.clp")
CLIPS> $***
TRUE
CLIPS> (reset)
CLIPS> (facts)
f-0     (initial-fact)
f-1     (prioridad primera)
f-2     (prioridad segunda)
f-3     (prioridad tercera)
For a total of 4 facts.
CLIPS> (rules)
regla-1
regla-2
regla-3
For a total of 3 defrules.
CLIPS> (agenda)
30     regla-1: f-1
20     regla-2: f-2
10     regla-3: f-3
For a total of 3 activations.
CLIPS> (run)
Escribe primera
Escribe segunda
Escribe tercera
CLIPS>
```

Prioridades

- Sintaxis:
  - `(declare (salience <numero>))`
- Valores:
  - Mínimo: -10000
  - Máximo: 10000
  - Defecto: 0
- Inconvenientes:
  - Abuso.
  - Contradicción con el objetivo de los sistemas basados en reglas.

Reglas de control

- Reglas de cambio de fases `ej-3.clp`

```
(defrule deteccion-a-aislamiento
  (declare (salience -10))
  ?fase <- (fase deteccion)
  =>
  (retract ?fase)
  (assert (fase aislamiento)))

(defrule aislamiento-a-recuperacion
  (declare (salience -10))
  ?fase <- (fase aislamiento)
  =>
  (retract ?fase)
  (assert (fase recuperacion)))

(defrule recuperacion-a-deteccion
  (declare (salience -10))
  ?fase <- (fase recuperacion)
  =>
  (retract ?fase)
  (assert (fase deteccion)))

(defrule detecta-fuego
  (fase deteccion)
  (luz-a roja)
  =>
  (assert (problema fuego)))

(deffacts inicio
  (fase deteccion) (luz-a roja))
```

Reglas de control

- **Sesión**

```
CLIPS> (load "ej-3.clp")
CLIPS> ****$
TRUE
CLIPS> (watch rules)
CLIPS> (reset)
CLIPS> (run 7)
FIRE    1 detecta-fuego: f-1,f-2
FIRE    2 deteccion-a-aislamiento: f-1
FIRE    3 aislamiento-a-recuperacion: f-4
FIRE    4 recuperacion-a-deteccion: f-5
FIRE    5 detecta-fuego: f-6,f-2
FIRE    6 deteccion-a-aislamiento: f-6
FIRE    7 aislamiento-a-recuperacion: f-7
CLIPS>
```

Reglas de control

- Cambio de fase mediante una regla `ej-4.clp`

```
(deffacts control
  (fase deteccion)
  (siguiente-fase deteccion aislamiento)
  (siguiente-fase aislamiento recuperacion)
  (siguiente-fase recuperacion deteccion))

(defrule cambio-de-fase
  (declare (salience -10))
  ?fase <- (fase ?actual)
  (siguiente-fase ?actual ?siguiente)
  =>
  (retract ?fase)
  (assert (fase ?siguiente)))

(defrule detecta-fuego
  (fase deteccion)
  (luz-a roja)
  =>
  (assert (problema fuego)))

(deffacts inicio
  (fase deteccion)
  (luz-a roja))
```

Reglas de control

```
CLIPS> (load "ej-4.clp")
$**$
TRUE
CLIPS> (watch rules)
CLIPS> (watch facts)
CLIPS> (reset)
CLIPS> (reset)
==> f-0     (initial-fact)
==> f-1     (fase deteccion)
==> f-2     (siguiente-fase deteccion aislamiento)
==> f-3     (siguiente-fase aislamiento recuperacion)
==> f-4     (siguiente-fase recuperacion deteccion)
==> f-5     (luz-a roja)
CLIPS> (run 5)
FIRE    1 detecta-fuego: f-1,f-5
==> f-6     (problema fuego)
FIRE    2 cambio-de-fase: f-1,f-2
<== f-1     (fase deteccion)
==> f-7     (fase aislamiento)
FIRE    3 cambio-de-fase: f-7,f-3
<== f-7     (fase aislamiento)
==> f-8     (fase recuperacion)
FIRE    4 cambio-de-fase: f-8,f-4
<== f-8     (fase recuperacion)
==> f-9     (fase deteccion)
FIRE    5 detecta-fuego: f-9,f-5
```

Reglas de control

- Cambio de fase mediante una regla y sucesión de fases `ej-5.clp`

```
(deffacts control
  (fase deteccion)
  (sucesion-de-fases aislamiento
                     recuperacion
                     deteccion))

(defrule cambio-de-fase
  (declare (salience -10))
  ?fase <- (fase ?actual)
  (sucesion-de-fases ?siguiente $?resto)
  =>
  (retract ?fase)
  (assert (fase ?siguiente))
  (assert (sucesion-de-fases ?resto ?siguiente)))
```

Reglas de control

```
CLIPS> (clear)
CLIPS> (load "ej-5.clp")
$**$
TRUE
CLIPS> (watch facts)
CLIPS> (watch rules)
CLIPS> (reset)
==> f-0     (initial-fact)
==> f-1     (fase deteccion)
==> f-2     (sucesion-de-fases
             aislamiento recuperacion deteccion)
==> f-3     (luz-a roja)
CLIPS> (run 5)
FIRE    1 detecta-fuego: f-1,f-3
==> f-4     (problema fuego)
FIRE    2 cambio-de-fase: f-1,f-2
<== f-1     (fase deteccion)
==> f-5     (fase aislamiento)
==> f-6     (sucesion-de-fases
             recuperacion deteccion aislamiento)
FIRE    3 cambio-de-fase: f-5,f-6
<== f-5     (fase aislamiento)
==> f-7     (fase recuperacion)
==> f-8     (sucesion-de-fases
             deteccion aislamiento recuperacion)
FIRE    4 cambio-de-fase: f-7,f-8
<== f-7     (fase recuperacion)
==> f-9     (fase deteccion)
FIRE    5 detecta-fuego: f-9,f-3
```

Definición de reglas en módulos

```
CLIPS> (defmodule DETECCION)
CLIPS> (defmodule AISLAMIENTO)
CLIPS> (defmodule RECUPERACION)
CLIPS> (defrule detecta-fuego
         (luz-a roja)
         =>
         (assert (problema fuego)))
CLIPS> (ppdefrule detecta-fuego)
(defrule RECUPERACION::detecta-fuego
   (luz-a roja)
   =>
   (assert (problema fuego)))
CLIPS> (undefrule detecta-fuego)
CLIPS> (defrule DETECCION::detecta-fuego
         (luz-a roja)
         =>
         (assert (problema fuego)))
CLIPS> (ppdefrule detecta-fuego)
(defrule DETECCION::detecta-fuego
   (luz-a roja)
   =>
   (assert (problema fuego)))
CLIPS> (get-current-module)
DETECCION
CLIPS> (set-current-module AISLAMIENTO)
DETECCION
CLIPS> (get-current-module)
AISLAMIENTO
```

Lista de reglas en módulos

```
CLIPS> (list-defrules)
CLIPS> (set-current-module DETECCION)
AISLAMIENTO
CLIPS> (list-defrules)
detecta-fuego
For a total of 1 defrule.
CLIPS> (set-current-module AISLAMIENTO)
DETECCION
CLIPS> (list-defrules DETECCION)
detecta-fuego
For a total of 1 defrule.
CLIPS> (list-defrules *)
MAIN:
DETECCION:
   detecta-fuego
AISLAMIENTO:
RECUPERACION:
For a total of 1 defrule.
```

- Comentarios:
  - Otras constructores: `deftemplate`, `deffactt`.
  - Posibilidad de reglas con el mismo nombre en módulos distintos.

Definición de hechos en módulos

```
CLIPS> (deftemplate DETECCION::fallo
         (slot componente))
CLIPS> (assert (fallo (componente A)))
<Fact-0>
CLIPS> (facts)
f-0     (fallo (componente A))
For a total of 1 fact.
CLIPS> (deftemplate AISLAMIENTO::posible-fallo
         (slot componente))
CLIPS> (assert (posible-fallo (componente B)))
<Fact-1>
CLIPS> (facts)
f-1     (posible-fallo (componente B))
For a total of 1 fact.
CLIPS> (set-current-module DETECCION)
AISLAMIENTO
CLIPS> (facts)
f-0     (fallo (componente A))
For a total of 1 fact.
CLIPS> (facts AISLAMIENTO)
f-1     (posible-fallo (componente B))
For a total of 1 fact.
CLIPS> (facts *)
f-0     (fallo (componente A))
f-1     (posible-fallo (componente B))
For a total of 2 facts.
```

Exportación e importación de hechos

```
CLIPS> (defmodule DETECCION
         (export deftemplate fallo))
CLIPS> (deftemplate DETECCION::fallo
         (slot componente))
CLIPS> (defmodule AISLAMIENTO
         (export deftemplate posible-fallo))
CLIPS> (deftemplate AISLAMIENTO::posible-fallo
         (slot componente))
CLIPS> (defmodule RECUPERACION
         (import DETECCION deftemplate fallo)
         (import AISLAMIENTO deftemplate posible-fallo))
CLIPS> (deffacts DETECCION::inicio
         (fallo (componente A)))
CLIPS> (deffacts AISLAMIENTO::inicio
         (posible-fallo (componente B)))
CLIPS> (deffacts RECUPERACION::inicio
         (fallo (componente C))
         (posible-fallo (componente D)))
CLIPS> (reset)
CLIPS> (facts DETECCION)
f-1     (fallo (componente A))
f-3     (fallo (componente C))
For a total of 2 facts.
CLIPS> (facts AISLAMIENTO)
f-2     (posible-fallo (componente B))
f-4     (posible-fallo (componente D))
For a total of 2 facts.
CLIPS> (facts RECUPERACION)
f-1     (fallo (componente A))
f-2     (posible-fallo (componente B))
f-3     (fallo (componente C))
f-4     (posible-fallo (componente D))
For a total of 4 facts.
```

Agendas de módulos

```
CLIPS> (defrule DETECCION::regla-1
         (fallo (componente A | C))
         =>)
CLIPS> (defrule AISLAMIENTO::regla-2
         (posible-fallo (componente B | D))
         =>)
CLIPS> (defrule RECUPERACION::regla-3
         (fallo (componente A | C))
         (posible-fallo (componente B | D))
         =>)
CLIPS> (get-current-module)
RECUPERACION
CLIPS> (facts)
f-1     (fallo (componente A))
f-2     (posible-fallo (componente B))
f-3     (fallo (componente C))
f-4     (posible-fallo (componente D))
For a total of 4 facts.
CLIPS> (agenda)
0      regla-3: f-1,f-4
0      regla-3: f-1,f-2
0      regla-3: f-3,f-4
0      regla-3: f-3,f-2
For a total of 4 activations.
CLIPS> (agenda DETECCION)
0      regla-1: f-3
0      regla-1: f-1
For a total of 2 activations.
CLIPS> (agenda AISLAMIENTO)
0      regla-2: f-4
0      regla-2: f-2
For a total of 2 activations.
```

Agendas de módulos

```
CLIPS> (agenda *)
MAIN:
DETECCION:
   0      regla-1: f-3
   0      regla-1: f-1
AISLAMIENTO:
   0      regla-2: f-4
   0      regla-2: f-2
RECUPERACION:
   0      regla-3: f-1,f-4
   0      regla-3: f-1,f-2
   0      regla-3: f-3,f-4
   0      regla-3: f-3,f-2
For a total of 8 activations.
```

Pila de focos

```
CLIPS> (unwatch all)
CLIPS> (watch rules)
CLIPS> (run)
CLIPS> (focus DETECCION)
TRUE
CLIPS> (run)
FIRE    1 regla-1: f-3
FIRE    2 regla-1: f-1
CLIPS> (focus AISLAMIENTO)
TRUE
CLIPS> (focus RECUPERACION)
TRUE
CLIPS> (list-focus-stack)
RECUPERACION
AISLAMIENTO
CLIPS> (run)
FIRE    1 regla-3: f-1,f-4
FIRE    2 regla-3: f-1,f-2
FIRE    3 regla-3: f-3,f-4
FIRE    4 regla-3: f-3,f-2
FIRE    5 regla-2: f-4
FIRE    6 regla-2: f-2
CLIPS> (list-focus-stack)
CLIPS> (reset)
CLIPS> (focus AISLAMIENTO RECUPERACION)
TRUE
CLIPS> (list-focus-stack)
AISLAMIENTO
RECUPERACION
MAIN
```

Pila de focos

```
CLIPS> (run)
FIRE    1 regla-2: f-4
FIRE    2 regla-2: f-2
FIRE    3 regla-3: f-1,f-4
FIRE    4 regla-3: f-3,f-4
FIRE    5 regla-3: f-3,f-2
FIRE    6 regla-3: f-1,f-2
CLIPS> (reset)
CLIPS> (focus DETECCION DETECCION AISLAMIENTO DETECCION)
TRUE
CLIPS> (list-focus-stack)
DETECCION
AISLAMIENTO
DETECCION
MAIN
CLIPS> (run)
FIRE    1 regla-1: f-3
FIRE    2 regla-1: f-1
FIRE    3 regla-2: f-4
FIRE    4 regla-2: f-2
```

Procedimientos sobre la pila de focos

```
CLIPS> (list-focus-stack)
CLIPS> (focus RECUPERACION AISLAMIENTO DETECCION)
TRUE
CLIPS> (list-focus-stack)
RECUPERACION
AISLAMIENTO
DETECCION
CLIPS> (get-focus-stack)
(RECUPERACION AISLAMIENTO DETECCION)
CLIPS> (get-focus)
RECUPERACION
CLIPS> (pop-focus)
RECUPERACION
CLIPS> (get-focus-stack)
(AISLAMIENTO DETECCION)
CLIPS> (clear-focus-stack)
CLIPS> (get-focus-stack)
()
CLIPS> (pop-focus)
FALSE
CLIPS> (get-focus)
FALSE
```

Vigilancia de los focos

```
CLIPS> (reset)
CLIPS> (watch focus)
CLIPS> (focus DETECCION AISLAMIENTO RECUPERACION)
==> Focus RECUPERACION from MAIN
==> Focus AISLAMIENTO from RECUPERACION
==> Focus DETECCION from AISLAMIENTO
TRUE
CLIPS> (agenda *)
MAIN:
DETECCION:
   0      regla-1: f-3
   0      regla-1: f-1
AISLAMIENTO:
   0      regla-2: f-4
   0      regla-2: f-2
RECUPERACION:
   0      regla-3: f-1,f-4
   0      regla-3: f-3,f-4
   0      regla-3: f-3,f-2
   0      regla-3: f-1,f-2
For a total of 8 activations.
CLIPS> (run)
FIRE    1 regla-1: f-3
FIRE    2 regla-1: f-1
<== Focus DETECCION to AISLAMIENTO
FIRE    3 regla-2: f-4
FIRE    4 regla-2: f-2
<== Focus AISLAMIENTO to RECUPERACION
FIRE    5 regla-3: f-1,f-4
FIRE    6 regla-3: f-3,f-4
FIRE    7 regla-3: f-3,f-2
FIRE    8 regla-3: f-1,f-2
<== Focus RECUPERACION to MAIN
<== Focus MAIN
```

Focos en reglas

```
CLIPS> (clear)
CLIPS> (unwatch all)
CLIPS> (defmodule MAIN (export ?ALL))
CLIPS> (defrule MAIN::regla-1
         =>
         (printout t "Disparada regla-1 de MAIN" crlf)
         (focus B A))
CLIPS> (defmodule A
         (import MAIN deftemplate initial-fact))
CLIPS> (defrule A::regla-2
         =>
         (printout t "Disparada regla-2 de A" crlf))
CLIPS> (defmodule B
         (import MAIN ?ALL))
CLIPS> (defrule B::regla-3
         =>
         (printout t "Disparada regla-3 de B" crlf))
CLIPS> (defrule B::regla-4
         =>
         (printout t "Disparada regla-4 de B" crlf))
CLIPS> (reset)
CLIPS> (run)
Disparada regla-1 de MAIN
Disparada regla-3 de B
Disparada regla-4 de B
Disparada regla-2 de A
```

La acción return

```
CLIPS> (clear)
CLIPS> (defmodule MAIN (export ?ALL))
CLIPS> (defrule MAIN::regla-1
         =>
         (printout t "Disparada regla-1 de MAIN" crlf)
         (focus B A))
CLIPS> (defmodule A (import MAIN ?ALL))
CLIPS> (defrule A::regla-2
         =>
         (printout t "Disparada regla-2 de A" crlf))
CLIPS> (defmodule B (import MAIN ?ALL))
CLIPS> (defrule B::regla-3
         =>
         (return)
         (printout t "Disparada regla-3 de B" crlf))
CLIPS> (defrule B::regla-4
         =>
         (printout t "Disparada regla-4 de B" crlf))
CLIPS> (reset)
CLIPS> (run)
Disparada regla-1 de MAIN
Disparada regla-2 de A
CLIPS> (undefrule A::regla-2)
CLIPS> (defrule A::regla-2
         =>
         (printout t "Disparada regla-2 de A" crlf)
         (focus B))
CLIPS> (reset)
CLIPS> (run)
Disparada regla-1 de MAIN
Disparada regla-2 de A
Disparada regla-4 de B
```

Enfoque automático

```
CLIPS> (clear)
CLIPS> (defmodule MAIN (export ?ALL))
CLIPS> (defrule MAIN::regla-1
         =>
         (printout t "Disparada regla-1 de MAIN" crlf))
CLIPS> (defmodule A (import MAIN ?ALL))
CLIPS> (defrule A::regla-2
         (declare (auto-focus TRUE))
         =>
         (printout t "Disparada regla-2 de A" crlf))
CLIPS> (defrule A::regla-3
         =>
         (printout t "Disparada regla-3 de A" crlf))
CLIPS> (reset)
CLIPS> (agenda *)
MAIN:
   0      regla-1: f-0
A:
   0      regla-2: f-0
   0      regla-3: f-0
For a total of 3 activations.
CLIPS> (get-focus-stack)
(A MAIN)
CLIPS> (run)
Disparada regla-2 de A
Disparada regla-3 de A
Disparada regla-1 de MAIN
```

Control mediante módulos

```
CLIPS> (clear)
CLIPS> (load "ej-6.clp")
$*+*$++
TRUE
CLIPS> (reset)
CLIPS> (agenda *)
MAIN:
   0      cambio-de-fase: f-1
DETECCION:
   0      detecta-fuego: f-2
AISLAMIENTO:
RECUPERACION:
For a total of 2 activations.
CLIPS> (watch rules)
CLIPS> (watch focus)
CLIPS> (run 5)
FIRE    1 cambio-de-fase: f-1
==> Focus DETECCION from MAIN
FIRE    2 detecta-fuego: f-2
<== Focus DETECCION to MAIN
FIRE    3 cambio-de-fase: f-3
==> Focus AISLAMIENTO from MAIN
<== Focus AISLAMIENTO to MAIN
FIRE    4 cambio-de-fase: f-5
==> Focus RECUPERACION from MAIN
<== Focus RECUPERACION to MAIN
FIRE    5 cambio-de-fase: f-6
==> Focus DETECCION from MAIN
<== Focus DETECCION to MAIN
CLIPS>
```

Nim con módulos

- Módulo MAIN

```
(defmodule MAIN
  (export deftemplate numero-de-piezas
                      turno
                      initial-fact))

(defrule MAIN::inicio
  =>
  (focus INICIO))

(defrule MAIN::la-computadora-elige
  ?turno <- (turno c)
  (numero-de-piezas ~0)
  =>
  (focus COMPUTADORA)
  (retract ?turno)
  (assert (turno h)))

(defrule MAIN::el-humano-elige
  ?turno <- (turno h)
  (numero-de-piezas ~0)
  =>
  (focus HUMANO)
  (retract ?turno)
  (assert (turno c)))
```

Nim con módulos

- Módulo INICIO

```
(defmodule INICIO
  (import MAIN deftemplate numero-de-piezas
                           turno
                           initial-fact))

(defrule INICIO::elige-jugador
  (not (turno ?))
  =>
  (printout t "Elige quien empieza: "
              "computadora o Humano (c/h) ")
  (assert (turno (read))))

(defrule INICIO::incorrecta-eleccion-de-jugador
  ?eleccion <- (turno ?jugador&~c&~h)
  =>
  (retract ?eleccion)
  (printout t ?jugador " es distinto de c y h" crlf))

(defrule INICIO::elige-numero-de-piezas
  (not (numero-de-piezas-elegidas ?))
  =>
  (printout t "Escribe el numero de piezas: ")
  (assert (numero-de-piezas-elegidas (read))))
```

Nim con módulos

```
(defrule INICIO::incorrecta-eleccion-del-numero-de-piezas
  ?e <- (numero-de-piezas-elegidas ?n&~:(integerp ?n)
                                   |:(<= ?n 0))
  =>
  (retract ?e)
  (printout t ?n " no es un numero entero mayor que 0"
              crlf))

(defrule INICIO::correcta-eleccion-del-numero-de-piezas
  (numero-de-piezas-elegidas ?n&:(integerp ?n)
                               &:(> ?n 0))
  =>
  (assert (numero-de-piezas ?n))
  (return))
```

- Módulo HUMANO

```
(defmodule HUMANO
  (import MAIN deftemplate numero-de-piezas))

(defrule HUMANO::pierde-el-humano
  ?h <- (numero-de-piezas 1)
  =>
  (printout t "Tienes que coger la ultima pieza" crlf)
  (printout t "Has perdido" crlf)
  (retract ?h)
  (assert (numero-de-piezas 0)))
```

Nim con módulos

```
(defrule HUMANO::eleccion-humana
  (numero-de-piezas ?n&:(> ?n 1))
  (not (piezas-cogidas ?))
  =>
  (printout t "Escribe el numero de piezas que coges: ")
  (assert (piezas-cogidas (read))))

(defrule HUMANO::correcta-eleccion-humana
  ?pila <- (numero-de-piezas ?n)
  ?eleccion <- (piezas-cogidas ?m)
  (test (and (integerp ?m)
             (>= ?m 1)
             (<= ?m 3)
             (< ?m ?n)))
  =>
  (retract ?pila ?eleccion)
  (bind ?nuevo-numero-de-piezas (- ?n ?m))
  (assert (numero-de-piezas ?nuevo-numero-de-piezas))
  (printout t "Quedan " ?nuevo-numero-de-piezas
              " pieza(s)" crlf)
  (return))

(defrule HUMANO::incorrecta-eleccion-humana
  (numero-de-piezas ?n)
  ?eleccion <- (piezas-cogidas ?m)
  (test (or (not (integerp ?m))
            (< ?m 1)
            (> ?m 3)
            (>= ?m ?n)))
  =>
  (printout t "Tiene que elegir un numero entre 1 y 3"
            crlf)
  (retract ?eleccion))
```

Nim con módulos

- Módulo COMPUTADORA

```
(defmodule COMPUTADORA
  (import MAIN deftemplate numero-de-piezas))

(defrule COMPUTADORA::pierde-la-computadora
  ?h <- (numero-de-piezas 1)
  =>
  (printout t "La computadora coge la ultima pieza" crlf)
  (printout t "He perdido" crlf)
  (retract ?h)
  (assert (numero-de-piezas 0)))

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

Nim con módulos

- **Sesión:**

```
CLIPS> (clear)
CLIPS> (load "nim-con-modulos.clp")
+**************+*$*
TRUE
CLIPS> (reset)
CLIPS> (run)
Elige quien empieza: computadora o Humano (c/h) a
a es distinto de c y h
Elige quien empieza: computadora o Humano (c/h) c
Escribe el numero de piezas: -3
-3 no es un numero entero mayor que 0
Escribe el numero de piezas: a
a no es un numero entero mayor que 0
Escribe el numero de piezas: 5
La computadora coge 1 pieza(s)
Quedan 4 pieza(s)
Escribe el numero de piezas que coges: 4
Tiene que elegir un numero entre 1 y 3
Escribe el numero de piezas que coges: 1
Quedan 3 pieza(s)
La computadora coge 2 pieza(s)
Quedan 1 pieza(s)
Tienes que coger la ultima pieza
Has perdido
```

Nim con módulos

- Sesión con traza:

```
CLIPS> (watch facts)
CLIPS> (watch rules)
CLIPS> (watch focus)
CLIPS> (reset)
==> Focus MAIN
==> f-0     (initial-fact)
==> f-1     (computadora-coge 1 cuando-el-resto-es 1)
==> f-2     (computadora-coge 1 cuando-el-resto-es 2)
==> f-3     (computadora-coge 2 cuando-el-resto-es 3)
==> f-4     (computadora-coge 3 cuando-el-resto-es 0)
CLIPS> (run)
FIRE    1 inicio: f-0
==> Focus INICIO from MAIN
FIRE    2 elige-jugador: f-0,
Elige quien empieza: computadora o Humano (c/h) a
==> f-5     (turno a)
FIRE    3 incorrecta-eleccion-de-jugador: f-5
<== f-5     (turno a)
a es distinto de c y h
FIRE    4 elige-jugador: f-0,
Elige quien empieza: computadora o Humano (c/h) c
==> f-6     (turno c)
FIRE    5 elige-numero-de-piezas: f-0,
Escribe el numero de piezas: a
==> f-7     (numero-de-piezas-elegidas a)
FIRE    6 incorrecta-eleccion-del-numero-de-piezas: f-7
<== f-7     (numero-de-piezas-elegidas a)
a no es un numero entero mayor que 0
```

Nim con módulos

```
FIRE    7 elige-numero-de-piezas: f-0,
Escribe el numero de piezas: 5
==> f-8     (numero-de-piezas-elegidas 5)
FIRE    8 correcta-eleccion-del-numero-de-piezas: f-8
==> f-9     (numero-de-piezas 5)
<== Focus INICIO to MAIN
FIRE    9 la-computadora-elige: f-6,f-9
==> Focus COMPUTADORA from MAIN
<== f-6     (turno c)
==> f-10    (turno h)
FIRE   10 eleccion-computadora: f-9,f-1
<== f-9     (numero-de-piezas 5)
La computadora coge 1 pieza(s)
Quedan 4 pieza(s)
==> f-11    (numero-de-piezas 4)
<== Focus COMPUTADORA to MAIN
FIRE   11 el-humano-elige: f-10,f-11
==> Focus HUMANO from MAIN
<== f-10    (turno h)
==> f-12    (turno c)
FIRE   12 eleccion-humana: f-11,
Escribe el numero de piezas que coges: 4
==> f-13    (piezas-cogidas 4)
FIRE   13 incorrecta-eleccion-humana: f-11,f-13
Tiene que elegir un numero entre 1 y 3
<== f-13    (piezas-cogidas 4)
FIRE   14 eleccion-humana: f-11,
Escribe el numero de piezas que coges: 1
==> f-14    (piezas-cogidas 1)
```

Nim con módulos

```
FIRE   15 correcta-eleccion-humana: f-11,f-14
<== f-11    (numero-de-piezas 4)
<== f-14    (piezas-cogidas 1)
==> f-15    (numero-de-piezas 3)
Quedan 3 pieza(s)
<== Focus HUMANO to MAIN
FIRE   16 la-computadora-elige: f-12,f-15
==> Focus COMPUTADORA from MAIN
<== f-12    (turno c)
==> f-16    (turno h)
FIRE   17 eleccion-computadora: f-15,f-3
<== f-15    (numero-de-piezas 3)
La computadora coge 2 pieza(s)
Quedan 1 pieza(s)
==> f-17    (numero-de-piezas 1)
<== Focus COMPUTADORA to MAIN
FIRE   18 el-humano-elige: f-16,f-17
==> Focus HUMANO from MAIN
<== f-16    (turno h)
==> f-18    (turno c)
FIRE   19 pierde-el-humano: f-17
Tienes que coger la ultima pieza
Has perdido
<== f-17    (numero-de-piezas 1)
==> f-19    (numero-de-piezas 0)
<== Focus HUMANO to MAIN
<== Focus MAIN
```