

Clips: Resolución de problemas

José A. Alonso y Francisco J. Martín

Ciencias de la Computación e Inteligencia Artificial

UNIVERSIDAD DE SEVILLA

Mundo de los bloques (I)

- Enunciado

```
+----+   +----+
|  A  |   |  D  |
+----+   +----+
|  B  |   |  E  |
+----+   +----+
|  C  |   |  F  |
+----+   +----+
```

- Objetivo: Poner C encima de E

- Representación

```
(defacts estado-inicial
  (bloque A) (bloque B) (bloque C)
  (bloque D) (bloque E) (bloque F)
  (estado nada esta-encima-del A)
  (estado A esta-encima-del B)
  (estado B esta-encima-del C)
  (estado C esta-encima-del suelo)
  (estado nada esta-encima-del D)
  (estado D esta-encima-del E)
  (estado E esta-encima-del F)
  (estado F esta-encima-del suelo)
  (objetivo C esta-encima-del E))
```

Mundo de los bloques (I)

- Regla mover-bloque-sobre-bloque

```
;;; REGLA: mover-bloque-sobre-bloque
;;; SI
;;;   el objetivo es poner el objeto X encima del objeto Y y
;;;   tanto X como Y son bloques 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
  ?objetivo <- (objetivo ?objeto-1 esta-encima-del ?objeto-2)
  (bloque ?objeto-1)
  (bloque ?objeto-2)
  (estado nada esta-encima-del ?objeto-1)
  ?pila-1 <- (estado ?objeto-1 esta-encima-del ?objeto-3)
  ?pila-2 <- (estado nada esta-encima-del ?objeto-2)
  =>
  (retract ?objetivo ?pila-1 ?pila-2)
  (assert (estado ?objeto-1 esta-encima-del ?objeto-2))
  (assert (estado nada esta-encima-del ?objeto-3))
  (printout t ?objeto-1 " movido encima del "
            ?objeto-2 "." crlf))
```

Mundo de los bloques (I)

- Regla mover-bloque-al-suelo

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

```
(defrule mover-bloque-al-suelo
  ?objetivo <- (objetivo ?objeto-1 esta-encima-del suelo)
  (bloque ?objeto-1)
  (estado nada esta-encima-del ?objeto-1)
  ?pila <- (estado ?objeto-1 esta-encima-del ?objeto-2)
  =>
  (retract ?objetivo ?pila)
  (assert (estado ?objeto-1 esta-encima-del suelo))
  (assert (estado nada esta-encima-del ?objeto-2))
  (printout t ?objeto-1 " movido encima del suelo." crlf))
```

Mundo de los bloques (I)

- Regla libera-bloque-movible

```
;;; REGLA: libera-bloque-movible
;;; SI
;;;   el objetivo es poner el objeto 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 esta encima de X
;;;   en el suelo.
```

```
(defrule libera-bloque-movible
  (objetivo ?objeto-1 esta-encima-del ?)
  (bloque ?objeto-1)
  (estado ?objeto-2 esta-encima-del ?objeto-1)
  (bloque ?objeto-2)
  =>
  (assert (objetivo ?objeto-2 esta-encima-del suelo)))
```

Mundo de los bloques (I)

- Regla libera-bloque-soporte

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

```
(defrule libera-bloque-soporte
  (objetivo ? esta-encima-del ?objeto-1)
  (bloque ?objeto-1)
  (estado ?objeto-2 esta-encima-del ?objeto-1)
  (bloque ?objeto-2)
  =>
  (assert (objetivo ?objeto-2 esta-encima-del suelo)))
```

Mundo de los bloques (I)

```
CLIPS> (clear)
CLIPS> (load "bloques1.clp")
TRUE
CLIPS> (watch facts)
CLIPS> (watch rules)
CLIPS> (watch activations )
CLIPS> (reset)
==> f-0      (initial-fact)
==> f-1      (bloque A)
==> f-2      (bloque B)
==> f-3      (bloque C)
==> f-4      (bloque D)
==> f-5      (bloque E)
==> f-6      (bloque F)
==> f-7      (estado nada esta-encima-del A)
==> f-8      (estado A esta-encima-del B)
==> f-9      (estado B esta-encima-del C)
==> f-10     (estado C esta-encima-del suelo)
==> f-11     (estado nada esta-encima-del D)
==> f-12     (estado D esta-encima-del E)
==> f-13     (estado E esta-encima-del F)
==> f-14     (estado F esta-encima-del suelo)
==> f-15     (objetivo C esta-encima-del E)
```

Mundo de los bloques (I)

```
==> Activation 0      libera-bloque-soporte: f-15,f-5,f-12,f-4
==> Activation 0      libera-bloque-movible: f-15,f-3,f-9,f-2
CLIPS> (run)
FIRE    1 libera-bloque-movible: f-15,f-3,f-9,f-2
==> f-16      (objetivo B esta-encima-del suelo)
==> Activation 0      libera-bloque-movible: f-16,f-2,f-8,f-1
FIRE    2 libera-bloque-movible: f-16,f-2,f-8,f-1
==> f-17      (objetivo A esta-encima-del suelo)
==> Activation 0      mover-bloque-al-suelo: f-17,f-1,f-7,f-8
FIRE    3 mover-bloque-al-suelo: f-17,f-1,f-7,f-8
<== f-17      (objetivo A esta-encima-del suelo)
<== f-8       (estado A esta-encima-del B)
==> f-18      (estado A esta-encima-del suelo)
==> f-19      (estado nada esta-encima-del B)
==> Activation 0      mover-bloque-al-suelo: f-16,f-2,f-19,f-9
A movido encima del suelo.
FIRE    4 mover-bloque-al-suelo: f-16,f-2,f-19,f-9
<== f-16      (objetivo B esta-encima-del suelo)
<== f-9       (estado B esta-encima-del C)
==> f-20      (estado B esta-encima-del suelo)
==> f-21      (estado nada esta-encima-del C)
B movido encima del suelo.
FIRE    5 libera-bloque-soporte: f-15,f-5,f-12,f-4
==> f-22      (objetivo D esta-encima-del suelo)
==> Activation 0      mover-bloque-al-suelo: f-22,f-4,f-11,f-12
```


Mundo de los bloques (I)

```
FIRE      6 mover-bloque-al-suelo: f-22,f-4,f-11,f-12
<== f-22   (objetivo D esta-encima-del suelo)
<== f-12   (estado D esta-encima-del E)
==> f-23   (estado D esta-encima-del suelo)
==> f-24   (estado nada esta-encima-del E)
==> Activation 0      mover-bloque-sobre-bloque:
                        f-15,f-3,f-5,f-21,f-10,f-24
```

D movido encima del suelo.

```
FIRE      7 mover-bloque-sobre-bloque: f-15,f-3,f-5,f-21,f-10,f-24
<== f-15   (objetivo C esta-encima-del E)
<== f-10   (estado C esta-encima-del suelo)
<== f-24   (estado nada esta-encima-del E)
==> f-25   (estado C esta-encima-del E)
==> f-26   (estado nada esta-encima-del suelo)
```

C movido encima del E.

CLIPS>

Mundo de los bloques (II)

- Enunciado

```
+----+   +----+
| A |   | D |
+----+   +----+
| B |   | E |
+----+   +----+
| C |   | F |
+----+   +----+
```

- Objetivo: Poner C encima de E

- Representación

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

Mundo de los bloques (II)

- Reglas

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

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

Mundo de los bloques (II)

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

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

Mundo de los bloques (II)

● Sesión

```
CLIPS> (clear)
CLIPS> (load "bloques2.clp")
CLIPS> Defining deffacts: estado-inicial
Defining defrule: mover-bloque-sobre-bloque +j+j+j
Defining defrule: mover-bloque-al-suelo +j+j
Defining defrule: liberar-bloque-movible =j+j
Defining defrule: liberar-bloque-soporte =j+j
TRUE
CLIPS> (watch facts)
CLIPS> (watch rules)
CLIPS> (watch activations)
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 (II)

```
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 (III)

- Enunciado

```
+----+   +----+
|  A  |   |  D  |
+----+   +----+
|  B  |   |  E  |
+----+   +----+
|  C  |   |  F  |
+----+   +----+
```

- Objetivo: Poner C encima de E

- Representación

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

Mundo de los bloques (III)

● Reglas

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

```
(defrule mover-bloque-a-la-pila-auxiliar
  ?objetivo <- (mover ?bloque-1 a-la-pila-auxiliar)
  ?pila-1 <- (pila ?bloque-1 $?resto)
  ?pila-auxiliar <- (pila-auxiliar $?elementos)
  =>
  (retract ?objetivo ?pila-1 ?pila-auxiliar)
  (assert (pila-auxiliar ?bloque-1 $?elementos))
  (assert (pila $?resto))
  (printout t ?bloque-1 " movido a la pila auxiliar "
            $?elementos crlf))
```


Mundo de los bloques (III)

```
(defrule liberar-bloque-movible
  (objetivo ?bloque esta-encima-del ?)
  (pila ?cima $? ?bloque $?)
  =>
  (assert (mover ?cima a-la-pila-auxiliar)))
```

```
(defrule liberar-bloque-soporte
  (objetivo ? esta-encima-del ?bloque)
  (pila ?cima $? ?bloque $?)
  =>
  (assert (mover ?cima a-la-pila-auxiliar)))
```

Mundo de los bloques (III)

● Sesión

```
CLIPS> (clear)
CLIPS> (load "bloques3.clp")
CLIPS> Defining deffacts: estado-inicial
Defining defrule: mover-bloque-sobre-bloque +j+j+j
Defining defrule: mover-bloque-a-la-pila-auxiliar +j+j+j
Defining defrule: liberar-bloque-movible =j+j
Defining defrule: liberar-bloque-soporte =j+j
TRUE
CLIPS> (watch facts)
CLIPS> (watch rules)
CLIPS> (watch activations)
CLIPS> (reset)
==> f-0      (initial-fact)
==> f-1      (pila A B C)
==> f-2      (pila D E F)
==> f-3      (pila-auxiliar)
==> f-4      (objetivo C esta-encima-del E)
==> Activation 0      liberar-bloque-soporte: f-4,f-2
==> Activation 0      liberar-bloque-movible: f-4,f-1
CLIPS> (run)
FIRE      1 liberar-bloque-movible: f-4,f-1
==> f-5      (mover A a-la-pila-auxiliar)
==> Activation 0      mover-bloque-a-la-pila-auxiliar: f-5,f-
```

Mundo de los bloques (III)

```
FIRE      2 mover-bloque-a-la-pila-auxiliar: f-5,f-1,f-3
<== f-5      (mover A a-la-pila-auxiliar)
<== f-1      (pila A B C)
<== f-3      (pila-auxiliar)
==> f-6      (pila-auxiliar A)
==> f-7      (pila B C)
==> Activation 0      liberar-bloque-movible: f-4,f-7
A movido a la pila auxiliar ()
FIRE      3 liberar-bloque-movible: f-4,f-7
==> f-8      (mover B a-la-pila-auxiliar)
==> Activation 0      mover-bloque-a-la-pila-auxiliar: f-8,f-
FIRE      4 mover-bloque-a-la-pila-auxiliar: f-8,f-7,f-6
<== f-8      (mover B a-la-pila-auxiliar)
<== f-7      (pila B C)
<== f-6      (pila-auxiliar A)
==> f-9      (pila-auxiliar B A)
==> f-10     (pila C)
B movido a la pila auxiliar (A)
FIRE      5 liberar-bloque-soporte: f-4,f-2
==> f-11     (mover D a-la-pila-auxiliar)
==> Activation 0      mover-bloque-a-la-pila-auxiliar:
                        f-11,f-2,f-9
FIRE      6 mover-bloque-a-la-pila-auxiliar: f-11,f-2,f-9
<== f-11     (mover D a-la-pila-auxiliar)
<== f-2      (pila D E F)
<== f-9      (pila-auxiliar B A)
==> f-12     (pila-auxiliar D B A)
==> f-13     (pila E F)
==> Activation 0      mover-bloque-sobre-bloque: f-4,f-10,f-1
D movido a la pila auxiliar (B A)
```

Mundo de los bloques (III)

```
FIRE      7 mover-bloque-sobre-bloque: f-4,f-10,f-13
<== f-4    (objetivo C esta-encima-del E)
<== f-10   (pila C)
<== f-13   (pila E F)
==> f-14   (pila)
==> f-15   (pila C E F)
C movido encima del E
CLIPS>
```

Nim

● Sesión

```
CLIPS> (unwatch compilations)
CLIPS> ((load "nim.clp")
CLIPS> $*****$*
TRUE
CLIPS> (reset)
CLIPS> (run)
Elige quien empieza: computadora o Humano (c/h) c
Escribe el numero de piezas: 15
La computadora coge 2 pieza(s)
Quedan 13 pieza(s)
Escribe el numero de piezas que coges: 3
Quedan 10 pieza(s)
La computadora coge 1 pieza(s)
Quedan 9 pieza(s)
Escribe el numero de piezas que coges: 2
Quedan 7 pieza(s)
La computadora coge 2 pieza(s)
Quedan 5 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 4 pieza(s)
La computadora coge 3 pieza(s)
Quedan 1 pieza(s)
Tienes que coger la ultima pieza
Has perdido
CLIPS>
```

Nim

- Elección del jugador

```
(deffacts fase-inicial
  (fase elige-jugador))
```

```
(defrule elige-jugador
  (fase elige-jugador)
  =>
  (printout t "Elige quien empieza: ")
  (printout t "computadora o Humano (c/h) ")
  (assert (jugador-elegido (read))))
```

```
(defrule correcta-eleccion-de-jugador
  ?fase <- (fase elige-jugador)
  ?eleccion <- (jugador-elegido ?jugador&c|h)
  =>
  (retract ?fase ?eleccion)
  (assert (turno ?jugador))
  (assert (fase elige-numero-de-piezas)))
```

```
(defrule incorrecta-eleccion-de-jugador
  ?fase <- (fase elige-jugador)
  ?eleccion <- (jugador-elegido ?jugador&~c&~h)
  =>
  (retract ?fase ?eleccion)
  (printout t ?jugador " es distinto de c y h" crlf)
  (assert (fase elige-jugador)))
```

Nim

- Elección del número de piezas

```
(defrule elige-numero-de-piezas
  (fase elige-numero-de-piezas)
  =>
  (printout t "Escribe el numero de piezas: ")
  (assert (numero-de-piezas (read))))

(defrule correcta-eleccion-del-numero-de-piezas
  ?fase <- (fase elige-numero-de-piezas)
  ?eleccion <- (numero-de-piezas ?n&:(integerp ?n)
               &:(> ?n 0))
  =>
  (retract ?fase ?eleccion)
  (assert (numero-de-piezas ?n)))

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

Nim

- Jugada humana

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

```
(defrule eleccion-humana
  (turno h)
  (numero-de-piezas ?n&:(> ?n 1))
  =>
  (printout t "Escribe el numero de piezas que coges: ")
  (assert (piezas-cogidas (read))))
```


Nim

```
(defrule correcta-eleccion-humana
  ?pila <- (numero-de-piezas ?n)
  ?eleccion <- (piezas-cogidas ?m)
  ?turno <- (turno h)
  (test (and (integerp ?m)
             (>= ?m 1)
             (<= ?m 3)
             (< ?m ?n)))
  =>
  (retract ?pila ?eleccion ?turno)
  (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)
  (assert (turno c)))

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

Nim

- Jugada de la computadora

```
(defrule pierde-la-computadora
  (turno c)
  (numero-de-piezas 1)
  =>
  (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 "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))
  (assert (turno h)))
```