Problemas de espacios de estados

José A. Alonso y Francisco J. Martín

Ciencias de la Computación e Inteligencia Artificial

Universidad de Sevilla

Ordenación

- **Enunciado**
  - **Dada una lista de números obtener la lista ordenada de menor a mayor.**
- **Sesión**

```
CLIPS> (assert (vector 3 2 1 4))
La ordenacion de (3 2 1 4) es (1 2 3 4)
```

- **Solución**

```
(defrule inicial
  (vector $?x)
  =>
  (assert (vector-aux ?x)))

(defrule ordena
  ?f <- (vector-aux $?b ?m1 ?m2&:(< ?m2 ?m1) $?e)
  =>
  (retract ?f)
  (assert (vector-aux $?b ?m2 ?m1 $?e)))

(defrule final
  (not (vector-aux $?b ?m1 ?m2&:(< ?m2 ?m1) $?e))
  (vector $?x)
  (vector-aux $?y)
  =>
  (printout t "La ordenacion de " ?x " es " ?y crlf))
```

Numeración romana

• Enunciado

El siguiente conjunto de reglas lee un número x en notación decimal y, si está entre 1 y 3999, escribe su notación romana:

- **Si x es un número y $x > 3999$, entonces escribir "demasiado grande" y terminar.**
- **Si x es un número y $x = 0$, entonces terminar.**
- **Si x es un número y $1 \leq x \leq 3$, entonces escribir "I" y reducir x en 1.**
- **Si x es un número y $x = 4$, entonces escribir "IV" y terminar.**
- **Si x es un número y $5 \leq x \leq 8$, entonces escribir "V" y reducir x en 5.**
- **Si x es un número y $x = 9$, entonces escribir "IX" y terminar.**
- **Si x es un número y $10 \leq x \leq 39$, entonces escribir "X" y reducir x en 10.**
- **Si x es un número y $40 \leq x \leq 49$, entonces escribir "LX" y reducir x en 40.**
- **Si x es un número y $50 \leq x \leq 89$, entonces escribir "L" y reducir x en 50.**

Numeración romana

- **Enunciado**
  - **Si x es un número y $90 \leq x \leq 99$, entonces escribir "XC" y reducir x en 90.**
  - **Si x es un número y $100 \leq x \leq 399$, entonces escribir "C" y reducir x en 100.**
  - **Si x es un número y $400 \leq x \leq 499$, entonces escribir "CD" y reducir x en 400.**
  - **Si x es un número y $500 \leq x \leq 899$, entonces escribir "D" y reducir x en 500.**
  - **Si x es un número y $900 \leq x \leq 999$, entonces escribir "CM" y reducir x en 900.**
  - **Si x es un número y $1000 \leq x \leq 3999$, entonces escribir "M" y reducir x en 1000.**
- **Sesión**

```
CLIPS> (assert (numero 27))
<Fact-0>
CLIPS> (run)
XXVII
CLIPS> (assert (numero 1996))
<Fact-7>
CLIPS> (run)
MCMXCVI
```

Numeración romana

- **Base de conocimiento**

```
(defrule demasiado-grande
  ?h <- (numero ?x&:(> ?x 3999))
  =>
  (retract ?h)
  (printout t "El número " ?x
              " es demasiado grande." crlf))

(defrule inicial
  (numero ?x&:(<= ?x 3999))
  =>
  (assert (numero-aux ?x)))

(defrule reduce-0
  ?h <- (numero-aux 0)
  =>
  (retract ?h)
  (printout t "" crlf))

(defrule reduce-1
  ?h <- (numero-aux ?x&:(<= 1 ?x 3))
  =>
  (retract ?h)
  (printout t "I")
  (assert (numero-aux (- ?x 1))))
```

Numeración romana

- **Base de conocimiento**

```
(defrule reduce-4
  ?h <- (numero-aux 4)
  =>
  (retract ?h)
  (printout t "IV" crlf))

(defrule reduce-5
  ?h <- (numero-aux ?x&:(<= 5 ?x 8))
  =>
  (retract ?h)
  (printout t "V")
  (assert (numero-aux (- ?x 5))))

(defrule reduce-9
  ?h <- (numero-aux 9)
  =>
  (retract ?h)
  (printout t "IX" crlf))

(defrule reduce-10
  ?h <- (numero-aux ?x&:(<= 10 ?x 39))
  =>
  (retract ?h)
  (printout t "X")
  (assert (numero-aux (- ?x 10))))
```

Numeración romana

- **Base de conocimiento**

```
(defrule reduce-40
  ?h <- (numero-aux ?x&:(<= 40 ?x 49))
  =>
  (retract ?h)
  (printout t "LX")
  (assert (numero-aux (- ?x 40))))
(defrule reduce-50
  ?h <- (numero-aux ?x&:(<= 50 ?x 89))
  =>
  (retract ?h)
  (printout t "L")
  (assert (numero-aux (- ?x 50))))
(defrule reduce-90
  ?h <- (numero-aux ?x&:(<= 90 ?x 99))
  =>
  (retract ?h)
  (printout t "XC")
  (assert (numero-aux (- ?x 90))))
(defrule reduce-100
  ?h <- (numero-aux ?x&:(<= 100 ?x 399))
  =>
  (retract ?h)
  (printout t "C")
  (assert (numero-aux (- ?x 100))))
```

Numeración romana

- **Base de conocimiento**

```
(defrule reduce-400
  ?h <- (numero-aux ?x&:(<= 400 ?x 499))
  =>
  (retract ?h)
  (printout t "CD")
  (assert (numero-aux (- ?x 400))))
(defrule reduce-500
  ?h <- (numero-aux ?x&:(<= 500 ?x 899))
  =>
  (retract ?h)
  (printout t "D")
  (assert (numero-aux (- ?x 500))))
(defrule reduce-900
  ?h <- (numero-aux ?x&:(<= 900 ?x 999))
  =>
  (retract ?h)
  (printout t "CM")
  (assert (numero-aux (- ?x 900))))
(defrule reduce-1000
  ?h <- (numero-aux ?x&:(<= 1000 ?x 3999))
  =>
  (retract ?h)
  (printout t "M")
  (assert (numero-aux (- ?x 1000))))
```

Numeración romana

- **Traza**

```
CLIPS> (watch facts)
CLIPS> (watch rules)
CLIPS> (assert (numero 27))
==> f-1     (numero 27)
<Fact-1>
CLIPS> (run)
FIRE    1 inicial: f-1
==> f-2     (numero-aux 27)
FIRE    2 reduce-10: f-2
<== f-2     (numero-aux 27)
X
==> f-3     (numero-aux 17)
FIRE    3 reduce-10: f-3
<== f-3     (numero-aux 17)
X
==> f-4     (numero-aux 7)
FIRE    4 reduce-5: f-4
<== f-4     (numero-aux 7)
V
==> f-5     (numero-aux 2)
```

Numeración romana

- **Traza**

```
FIRE    5 reduce-1: f-5
<== f-5     (numero-aux 2)
I
==> f-6     (numero-aux 1)
FIRE    6 reduce-1: f-6
<== f-6     (numero-aux 1)
I
==> f-7     (numero-aux 0)
FIRE    7 reduce-0: f-7
<== f-7     (numero-aux 0)
```

Problema de las jarras

- **Enunciado**
  - **Se tienen dos jarras, una de 4 litros de capacidad y otra de 3.**
  - **Ninguna de ellas tiene marcas de medición.**
  - **Se tiene una bomba que permite llenar las jarras de agua.**
  - **Averiguar cómo se puede lograr tener exactamente 2 litros de agua en la jarra de 4 litros de capacidad?.**
- **Sesión**

```
CLIPS> (reset)
CLIPS> (run)

Solucion encontrada:
(llenar-jarra-4 llenar-jarra-3 vaciar-jarra-4
 vaciar-jarra-3-en-jarra-4 llenar-jarra-3
 llenar-jarra-4-con-jarra-3)
```

- **Representación de nodos**

```
(deftemplate nodo
  (multislot estado)
  (multislot camino))
```

Problema de las jarras

• Estado inicial

```
(deffacts nodo-inicial
  (nodo (estado 0 0)
        (camino)))
```

• Operadores

```
;;; (x,y) -> (4,y), si x < 4.
(defrule llenar-jarra-4
  (nodo (estado ?x&:(< ?x 4) ?y)
        (camino $?movimientos))
  (not (nodo (estado 4 ?y)))
  =>
  (assert (nodo (estado 4 ?y)
                (camino $?movimientos llenar-jarra-4))))

;;; (x,y) -> (x,3), si y < 3.
(defrule llenar-jarra-3
  (nodo (estado ?x ?y&:(< ?y 3))
        (camino $?movimientos))
  (not (nodo (estado ?x 3)))
  =>
  (assert (nodo (estado ?x 3)
                (camino $?movimientos llenar-jarra-3))))
```

Problema de las jarras

- **Operadores**

```
;;; (x,y) -> (0,y), si x > 0.
(defrule vaciar-jarra-4
  (nodo (estado ?x&:(> ?x 0) ?y)
        (camino $?movimientos))
  (not (nodo (estado 0 ?y)))
  =>
  (assert (nodo (estado 0 ?y)
                (camino $?movimientos vaciar-jarra-4))))

;;; (x,y) -> (x,0), si y > 0.
(defrule vaciar-jarra-3
  (nodo (estado ?x ?y&:(> ?y 0))
        (camino $?movimientos))
  (not (nodo (estado ?x 0)))
  =>
  (assert (nodo (estado ?x 0)
                (camino $?movimientos vaciar-jarra-3))))
```

Problema de las jarras

- **Operadores**

```
;;; (x,y) -> (4,y-(4-x)), si x < 4, y x + y > 4.
(defrule llenar-jarra-4-con-jarra-3
  (nodo (estado ?x&:(< ?x 4) ?y&:(> (+ ?x ?y) 4))
        (camino $?movimientos))
  (not (nodo (estado 4 =(- ?y (- 4 ?x)))))
  =>
  (assert (nodo (estado 4 (- ?y (- 4 ?x)))
                (camino $?movimientos
                        llenar-jarra-4-con-jarra-3))))

;;; (x,y) -> (x-(3-y),3), si y < 3 y x + y > 3.
(defrule llenar-jarra-3-con-jarra-4
  (nodo (estado ?x ?y&:(< ?y 3))
        (camino $?movimientos))
  (test (> (+ ?x ?y) 3))
  (not (nodo (estado =(- ?x (- 3 ?y)) 3)))
  =>
  (assert (nodo (estado (- ?x (- 3 ?y)) 3)
                (camino $?movimientos
                        llenar-jarra-3-con-jarra-4))))
```

Problema de las jarras

- **Operadores**

```
;;; (x,y) -> (x+y,0), si y > 0 y x + y <= 4.
(defrule vaciar-jarra-3-en-jarra-4
  (nodo (estado ?x ?y&:(> ?y 0))
        (camino $?movimientos))
  (test (<= (+ ?x ?y) 4))
  (not (nodo (estado =(+ ?x ?y) 0)))
  =>
  (assert (nodo (estado (+ ?x ?y) 0)
                (camino $?movimientos
                        vaciar-jarra-3-en-jarra-4))))

;;; (x,y) -> (0,x+y), si x > 0 y x + y <= 3.
(defrule vaciar-jarra-4-en-jarra-3
  (nodo (estado ?x&:(> ?x 0) ?y&:(<= (+ ?x ?y) 3))
        (camino $?movimientos))
  (not (nodo (estado 0 =(+ ?x ?y))))
  =>
  (assert (nodo (estado 0 (+ ?x ?y))
                (camino $?movimientos
                        vaciar-jarra-4-en-jarra-3))))
```

Problema de las jarras

- **Estados finales**

```
(defrule reconoce-solucion-1
  (declare (salience 100))
  ?nodo <- (nodo (estado 2 ?)
                 (camino $?movimientos))
  =>
  (retract ?nodo)
  (assert (solucion $?movimientos)))

(defrule reconoce-solucion-2
  (declare (salience 100))
  ?nodo <- (nodo (estado ? 2)
                 (camino $?movimientos))
  =>
  (retract ?nodo)
  (assert (solucion $?movimientos)))

(defrule escribe-solucion
  (declare (salience 100))
  (solucion $?m)
  =>
  (printout t crlf "Solucion encontrada: " crlf $?m crlf)
  (retract *))
```

Problema de las jarras

- **Traza con abreviaturas**

```
Abreviaturas de reglas:
L4   = llenar-jarra-4
L3   = llenar-jarra-3
V4   = vaciar-jarra-4
V3   = vaciar-jarra-3
L4C3 = llenar-jarra-4-con-jarra-3
L3C4 = llenar-jarra-3-con-jarra-4
V3E4 = vaciar-jarra-3-en-jarra-4
V4E3 = vaciar-jarra-4-en-jarra-3

Abreviaturas de representacion:
(nodo (estado ?x ?y) (camino $?z)) = ((?x ?y) ($?z))

CLIPS> (reset)
==> f-0     (initial-fact)
==> f-1     ((0 0) ())
==> Activation 0      L3: f-1,
==> Activation 0      L4: f-1,
CLIPS> (run)
FIRE    1 L4: f-1,
==> f-2     ((4 0) (L4))
==> Activation 0      L3C4: f-2,
==> Activation 0      L3: f-2,
FIRE    2 L3: f-2,
==> f-3     ((4 3) (L4 L3))
==> Activation 0      V4: f-3,
```

Problema de las jarras

- **Traza con abreviaturas**

```
FIRE    3 V4: f-3,
==> f-4     ((0 3) (L4 L3 V4))
==> Activation 0      V3E4: f-4,
<== Activation 0      L3: f-1,
FIRE    4 V3E4: f-4,
==> f-5     ((3 0) (L4 L3 V4 V3E4))
==> Activation 0      L3: f-5,
FIRE    5 L3: f-5,
==> f-6     ((3 3) (L4 L3 V4 V3E4 L3))
==> Activation 0      L4C3: f-6,
FIRE    6 L4C3: f-6,
==> f-7     ((4 2) (L4 L3 V4 V3E4 L3 L4C3))
==> Activation 0      V4: f-7,
==> Activation 100    reconoce-solucion-2: f-7
FIRE    7 reconoce-solucion-2: f-7
<== f-7     ((4 2) (L4 L3 V4 V3E4 L3 L4C3))
<== Activation 0      V4: f-7,
==> Activation 0      L4C3: f-6,
==> f-8     (solucion L4 L3 V4 V3E4 L3 L4C3)
==> Activation 100    escribe-solucion: f-8
FIRE    8 escribe-solucion: f-8
<== f-8     (solucion L4 L3 V4 V3E4 L3 L4C3)

Solucion encontrada:
(L4 L3 V4 V3E4 L3 L4C3)
```

Problema del granjero

- **Enunciado:**
  - **Un granjero está con un lobo, una cabra y una col en una orilla de un río.**
  - **Desea pasarlos a la otra orilla.**
  - **Dispone de una barca en la que sólo puede llevar una cosa cada vez.**
  - **El lobo se come a la cabra si no está el granjero.**
  - **La cabra se come la col si no está el granjero.**

Problema del granjero

- **Sesión**

```
CLIPS> (reset)
CLIPS> (run)

Solucion encontrada:
El granjero se mueve con la cabra a la derecha.
El granjero se mueve solo a la izquierda.
El granjero se mueve con el lobo a la derecha.
El granjero se mueve con la cabra a la izquierda.
El granjero se mueve con la col a la derecha.
El granjero se mueve solo a la izquierda.
El granjero se mueve con la cabra a la derecha.

Solucion encontrada:
El granjero se mueve con la cabra a la derecha.
El granjero se mueve solo a la izquierda.
El granjero se mueve con la col a la derecha.
El granjero se mueve con la cabra a la izquierda.
El granjero se mueve con el lobo a la derecha.
El granjero se mueve solo a la izquierda.
El granjero se mueve con la cabra a la derecha.
```

Problema del granjero

• Representación de estados

```
(deftemplate nodo
   (slot posicion-granjero)
   (slot posicion-lobo)
   (slot posicion-cabra)
   (slot posicion-col)
   (multislot camino))
```

• Estado inicial

```
(deffacts nodo-inicial
  (nodo (posicion-granjero izquierda)
        (posicion-lobo     izquierda)
        (posicion-cabra    izquierda)
        (posicion-col      izquierda)
        (camino)))
```

• Función auxiliar

```
(deffunction opuesta (?orilla)
  (if (eq ?orilla izquierda)
      then derecha
      else izquierda))
```

Problema del granjero

- **Operadores**

```
(defrule movimiento-solo
  ?nodo <- (nodo (posicion-granjero ?orilla-actual)
                 (camino $?movimientos))
  =>
  (duplicate ?nodo
             (posicion-granjero (opuesta ?orilla-actual))
             (camino $?movimientos solo)))

(defrule movimiento-con-lobo
  ?nodo <- (nodo (posicion-granjero ?orilla-actual)
                 (posicion-lobo ?orilla-actual)
                 (camino $?movimientos))
  =>
  (duplicate ?nodo
             (posicion-granjero (opuesta ?orilla-actual))
             (posicion-lobo (opuesta ?orilla-actual))
             (camino $?movimientos lobo)))
```

Problema del granjero

- **Operadores**

```
(defrule movimiento-con-cabra
  ?nodo <- (nodo (posicion-granjero ?orilla-actual)
                 (posicion-cabra ?orilla-actual)
                 (camino $?movimientos))
  =>
  (duplicate ?nodo
             (posicion-granjero (opuesta ?orilla-actual))
             (posicion-cabra (opuesta ?orilla-actual))
             (camino $?movimientos cabra)))

(defrule movimiento-con-col
  ?nodo <- (nodo (posicion-granjero ?orilla-actual)
                 (posicion-col ?orilla-actual)
                 (camino $?movimientos))
  =>
  (duplicate ?nodo
             (posicion-granjero (opuesta ?orilla-actual))
             (posicion-col (opuesta ?orilla-actual))
             (camino $?movimientos col)))
```

Problema del granjero

- **Restricciones**

```
(defrule lobo-come-cabra
  (declare (salience 200))
  ?nodo <- (nodo (posicion-granjero ?s1)
                 (posicion-lobo ?s2&~?s1)
                 (posicion-cabra ?s2))
  =>
  (retract ?nodo))

(defrule cabra-come-col
  (declare (salience 200))
  ?nodo <- (nodo (posicion-granjero ?s1)
                 (posicion-cabra ?s2&~?s1)
                 (posicion-col ?s2))
  =>
  (retract ?nodo))
```

Problema del granjero

- **Restricciones**

```
(defrule repeticion-de-estado
  (declare (salience 200))
  (nodo (posicion-granjero ?granjero)
        (posicion-lobo ?lobo)
        (posicion-cabra ?cabra)
        (posicion-col ?col)
        (camino $?movimientos-1))
  ?nodo <- (nodo (posicion-granjero ?granjero)
                 (posicion-lobo ?lobo)
                 (posicion-cabra ?cabra)
                 (posicion-col ?col)
                 (camino $?movimientos-1 ?
                         $?movimientos-2))
  =>
  (retract ?nodo))
```

Problema del granjero

- **Estado final y escritura de solución**

```
(defrule reconoce-solucion
  (declare (salience 100))
  ?nodo <- (nodo (posicion-granjero derecha)
                 (posicion-lobo derecha)
                 (posicion-cabra derecha)
                 (posicion-col derecha)
                 (camino $?movimientos))
  =>
  (retract ?nodo)
  (assert (solucion $?movimientos)))
```

Problema del granjero

- **Estado final y escritura de solución**

```
(defrule escribe-solucion
  (declare (salience 100))
  ?mv <- (solucion $?m)
  =>
  (retract ?mv)
  (printout t crlf "Solucion encontrada: " crlf)
  (bind ?orilla derecha)
  (loop-for-count (?i 1 (length $?m))
     (bind ?cosa (nth ?i $?m))
     (printout t "El granjero se mueve "
               (switch ?cosa
                       (case solo  then "solo ")
                       (case lobo  then "con el lobo ")
                       (case cabra then "con la cabra ")
                       (case col   then "con la col "))
               "a la " ?orilla "." crlf)
     (bind ?orilla (opuesta ?orilla))))
```

Problema del granjero con módulos

- **Módulo principal**

```
(defmodule MAIN
  (export deftemplate nodo)
  (export deffunction opuesta))

(deftemplate MAIN::nodo
   (slot posicion-granjero)
   (slot posicion-lobo)
   (slot posicion-cabra)
   (slot posicion-col)
   (multislot camino))

(deffacts MAIN::nodo-inicial
  (nodo (posicion-granjero izquierda)
        (posicion-lobo     izquierda)
        (posicion-cabra    izquierda)
        (posicion-col      izquierda)
        (camino)))

(deffunction MAIN::opuesta (?orilla)
  (if (eq ?orilla izquierda)
      then derecha
      else izquierda))
```

Problema del granjero con módulos

- **Módulo principal**

```
(defrule MAIN::movimiento-solo
  ?nodo <- (nodo (posicion-granjero ?orilla-actual)
                 (camino $?movimientos))
  =>
  (duplicate ?nodo
             (posicion-granjero (opuesta ?orilla-actual))
             (camino $?movimientos solo)))

(defrule MAIN::movimiento-con-lobo
  ?nodo <- (nodo (posicion-granjero ?orilla-actual)
                 (posicion-lobo ?orilla-actual)
                 (camino $?movimientos))
  =>
  (duplicate ?nodo
             (posicion-granjero (opuesta ?orilla-actual))
             (posicion-lobo (opuesta ?orilla-actual))
             (camino $?movimientos lobo)))
```

Problema del granjero con módulos

- **Módulo principal**

```
(defrule MAIN::movimiento-con-cabra
  ?nodo <- (nodo (posicion-granjero ?orilla-actual)
                 (posicion-cabra ?orilla-actual)
                 (camino $?movimientos))
  =>
  (duplicate ?nodo
             (posicion-granjero (opuesta ?orilla-actual))
             (posicion-cabra (opuesta ?orilla-actual))
             (camino $?movimientos cabra)))

(defrule MAIN::movimiento-con-col
  ?nodo <- (nodo (posicion-granjero ?orilla-actual)
                 (posicion-col ?orilla-actual)
                 (camino $?movimientos))
  =>
  (duplicate ?nodo
             (posicion-granjero (opuesta ?orilla-actual))
             (posicion-col (opuesta ?orilla-actual))
             (camino $?movimientos col)))
```

Problema del granjero con módulos

- **Módulo de restricciones**

```
(defmodule RESTRICCIONES
  (import MAIN deftemplate nodo))

(defrule RESTRICCIONES::lobo-come-cabra
  (declare (auto-focus TRUE))
  ?nodo <- (nodo (posicion-granjero ?s1)
                 (posicion-lobo ?s2&~?s1)
                 (posicion-cabra ?s2))
  =>
  (retract ?nodo))

(defrule RESTRICCIONES::cabra-come-col
  (declare (auto-focus TRUE))
  ?nodo <- (nodo (posicion-granjero ?s1)
                 (posicion-cabra ?s2&~?s1)
                 (posicion-col ?s2))
  =>
  (retract ?nodo))
```

Problema del granjero con módulos

- **Módulo de restricciones**

```
(defrule RESTRICCIONES::repeticion-de-nodo
  (declare (auto-focus TRUE))
  (nodo (posicion-granjero ?granjero)
        (posicion-lobo ?lobo)
        (posicion-cabra ?cabra)
        (posicion-col ?col)
        (camino $?movimientos-1))
  ?nodo <- (nodo (posicion-granjero ?granjero)
                 (posicion-lobo ?lobo)
                 (posicion-cabra ?cabra)
                 (posicion-col ?col)
                 (camino $?movimientos-1 ?
                         $?movimientos-2))
  =>
  (retract ?nodo))
```

Problema del granjero con módulos

- **Módulo solución**

```
(defmodule SOLUCION
  (import MAIN deftemplate nodo)
  (import MAIN deffunction opuesta))

(defrule SOLUCION::reconoce-solucion
  (declare (auto-focus TRUE))
  ?nodo <- (nodo (posicion-granjero derecha)
                 (posicion-lobo derecha)
                 (posicion-cabra derecha)
                 (posicion-col derecha)
                 (camino $?movimientos))
  =>
  (retract ?nodo)
  (assert (solucion $?movimientos)))
```

Problema del granjero con módulos

- **Módulo solución**

```
(defrule SOLUCION::escribe-solucion
  ?mv <- (solucion $?m)
  =>
  (retract ?mv)
  (printout t crlf "Solucion encontrada: " crlf)
  (bind ?orilla derecha)
  (loop-for-count (?i 1 (length $?m))
     (bind ?cosa (nth ?i $?m))
     (printout t "El granjero se mueve "
               (switch ?cosa
                       (case solo  then "solo ")
                       (case lobo  then "con el lobo ")
                       (case cabra then "con la cabra ")
                       (case col   then "con la col "))
               "a la " ?orilla "." crlf)
     (bind ?orilla (opuesta ?orilla))))
```

Problema del granjero con módulos

- **Traza**

```
CLIPS> (watch facts)
CLIPS> (watch rules)
CLIPS> (watch activations)
CLIPS> (watch focus)
CLIPS> (reset)
<== Focus MAIN
==> Focus MAIN
==> f-0     (initial-fact)
==> f-1     (nodo (posicion-granjero izquierda)
                  (posicion-lobo izquierda)
                  (posicion-cabra izquierda)
                  (posicion-col izquierda)
                  (camino))
==> Activation 0      movimiento-con-col: f-1
==> Activation 0      movimiento-con-cabra: f-1
==> Activation 0      movimiento-con-lobo: f-1
==> Activation 0      movimiento-solo: f-1
```

Problema del granjero con módulos

- **Traza**

```
CLIPS> (run 1)
FIRE    1 movimiento-solo: f-1
==> f-2     (nodo (posicion-granjero derecha)
                  (posicion-lobo izquierda)
                  (posicion-cabra izquierda)
                  (posicion-col izquierda)
                  (camino solo))
==> Focus RESTRICCIONES from MAIN
==> Activation 0      cabra-come-col: f-2
==> Activation 0      lobo-come-cabra: f-2
==> Activation 0      movimiento-solo: f-2
CLIPS> (get-focus-stack)
(RESTRICCIONES MAIN)
CLIPS> (agenda)
0      lobo-come-cabra: f-2
0      cabra-come-col: f-2
For a total of 2 activations.
```

Problema del granjero con módulos

- **Traza**

```
CLIPS> (run 1)
FRE    1 lobo-come-cabra: f-2
<== f-2     (nodo (posicion-granjero derecha)
                  (posicion-lobo izquierda)
                  (posicion-cabra izquierda)
                  (posicion-col izquierda)
                  (camino solo))
<== Activation 0      movimiento-solo: f-2
<== Activation 0      cabra-come-col: f-2
<== Focus RESTRICCIONES to MAIN
CLIPS> (get-focus-stack)
(MAIN)
CLIPS> (agenda)
0      movimiento-con-lobo: f-1
0      movimiento-con-cabra: f-1
0      movimiento-con-col: f-1
For a total of 3 activations.
```

Problema del granjero con módulos

- **Traza**

```
CLIPS> (unwatch all)
CLIPS> (run)

Solucion encontrada:
El granjero se mueve con la cabra a la derecha.
El granjero se mueve solo a la izquierda.
El granjero se mueve con el lobo a la derecha.
El granjero se mueve con la cabra a la izquierda.
El granjero se mueve con la col a la derecha.
El granjero se mueve solo a la izquierda.
El granjero se mueve con la cabra a la derecha.

Solucion encontrada:
El granjero se mueve con la cabra a la derecha.
El granjero se mueve solo a la izquierda.
El granjero se mueve con la col a la derecha.
El granjero se mueve con la cabra a la izquierda.
El granjero se mueve con el lobo a la derecha.
El granjero se mueve solo a la izquierda.
El granjero se mueve con la cabra a la derecha.
```

El problema de las fichas

- **Enunciado**
  - **La situación inicial es**

```
+---+---+---+---+---+---+---+
| B | B | B |   | V | V | V |
+---+---+---+---+---+---+---+
```

  - **La situación final es**

```
+---+---+---+---+---+---+---+
| V | V | V |   | B | B | B |
+---+---+---+---+---+---+---+
```

  - **Los movimientos permitidos consisten en desplazar una ficha al hueco saltando, como máximo, sobre otras dos.**

El problema de las fichas

- **Módulos principal**

```
(defmodule MAIN
  (export deftemplate nodo))

(deftemplate MAIN::nodo
  (multislot estado)
  (multislot camino))

(deffacts MAIN::nodo-inicial
  (nodo (estado B B B H V V V)
        (camino "B B B H V V V")))

(defrule MAIN::movimiento-izquierda
  ?nodo <- (nodo (estado $?x
                         H
                         $?y&:(<= (length $?y) 2)
                         ?ficha
                         $?z)
                 (camino $?movimientos))
  =>
  (bind $?nuevo-estado (create$ $?x ?ficha $?y H $?z))
  (duplicate ?nodo (estado $?nuevo-estado)
                   (camino $?movimientos
                           (implode$ $?nuevo-estado))))
```

El problema de las fichas

```
(defrule MAIN::movimiento-derecha
  ?nodo <- (nodo (estado $?x
                         ?ficha
                         $?y&:(<= (length $?y) 2)
                         H
                         $?z)
                 (camino $?movimientos))
  =>
  (bind $?nuevo-estado (create$ $?x H $?y ?ficha $?z))
  (duplicate ?nodo (estado $?nuevo-estado)
                   (camino $?movimientos
                           (implode$ $?nuevo-estado))))
```

- **Módulo de restricciones**

```
(defmodule RESTRICCIONES
  (import MAIN deftemplate nodo))

(defrule RESTRICCIONES::repeticion-de-nodo
  (declare (auto-focus TRUE))
  (nodo (estado $?actual)
        (camino $?movimientos-1))
  ?nodo <- (nodo (estado $?actual)
                 (camino $?movimientos-1 ? $?))
  =>
  (retract ?nodo))
```

El problema de las fichas

- **Módulo de reconocimiento y escritura de solución**

```
(defmodule SOLUCION
  (import MAIN deftemplate nodo))

(defrule SOLUCION::reconoce-solucion
  (declare (auto-focus TRUE))
  ?nodo <- (nodo (estado V V V H B B B)
                 (camino $?estados))
  =>
  (retract ?nodo)
  (assert (solucion $?estados)))

(defrule SOLUCION::escribe-solucion
  (solucion $?estados)
  =>
  (bind ?longitud (length $?estados))
  (printout t "La solucion, de longitud " ?longitud
              " es " crlf)
  (loop-for-count (?i 1 ?longitud)
    (bind ?estado (nth ?i $?estados))
    (printout t ?estado crlf))
  (retract *))
```

El problema de las fichas

- **Sesión con estadística**

```
CLIPS> (load "fichas-1.clp")
+%$*******
TRUE
CLIPS> (watch statistics)
CLIPS> (reset)
CLIPS> (run)
La solucion, de longitud 83 es:
B B B H V V V
H B B B V V V
B H B B V V V
B B H B V V V
B B V B H V V
.............
V B V V H B B
V H V V B B B
H V V V B B B
V V H V B B B
V V V H B B B
238 rules fired        Run time is 34.50 seconds.
6.898550724637682 rules per second.
43 mean number of facts (84 maximum).
1 mean number of instances (1 maximum).
103 mean number of activations (205 maximum).
```

El problema de las fichas con heurística

- **Heurística:**
  - **Definición: La heurística de un estado es la suma de piezas blancas situadas a la izquierda de cada una de las piezas verdes.**
  - **Ejemplo: La heurística del siguiente estado es 1+2+2 = 5.**

```
+---+---+---+---+---+---+---+
| B | V | B |   | V | V | B |
+---+---+---+---+---+---+---+
```

- **Módulos principal**

```
(defmodule MAIN
  (export deftemplate nodo))

(deftemplate MAIN::nodo
  (multislot estado)
  (multislot camino)
  (slot heuristica)
  (slot clase (default abierto)))

(defglobal MAIN
  ?*estado-inicial* = (create$ B B B H V V V))
```

El problema de las fichas con heurística

```
(deffunction heuristica ($?estado)
  (bind ?resultado 0)
  (bind ?verdes-restantes 3)
  (loop-for-count (?i 1 7)
    (bind ?ficha (nth ?i $?estado))
    (if (eq ?ficha B)
        then (bind ?resultado
                   (+ ?resultado ?verdes-restantes))
        else (if (eq ?ficha V)
                 then (bind ?verdes-restantes
                            (- ?verdes-restantes 1)))))
  ?resultado)

(defrule MAIN::inicial
  =>
  (assert (nodo (estado ?*estado-inicial*)
                (camino (implode$ ?*estado-inicial*))
                (heuristica (heuristica ?*estado-inicial*))
                (clase cerrado))))
```

El problema de las fichas con heurística

```
(defrule MAIN::movimiento-izquierda
  (nodo (estado $?x H
                $?y&:(<= (length $?y) 2)
                ?ficha $?z)
        (camino $?movimientos)
        (clase cerrado))
  =>
  (bind $?nuevo-estado (create$ $?x ?ficha $?y H $?z))
  (assert (nodo (estado $?nuevo-estado)
                (camino $?movimientos
                        (implode$ $?nuevo-estado))
                (heuristica (heuristica $?nuevo-estado)))))

(defrule MAIN::movimiento-derecha
  (nodo (estado $?x ?ficha
                $?y&:(<= (length $?y) 2)
                H $?z)
        (camino $?movimientos)
        (clase cerrado))
  =>
  (bind $?nuevo-estado (create$ $?x H $?y ?ficha $?z))
  (assert (nodo (estado $?nuevo-estado)
                (camino $?movimientos
                        (implode$ $?nuevo-estado))
                (heuristica (heuristica $?nuevo-estado)))))
```

El problema de las fichas con heurística

```
(defrule MAIN::pasa-el-mejor-a-cerrado
  (declare (salience -10))
  ?nodo <- (nodo (clase abierto)
                 (heuristica ?h1))
  (not (nodo (clase abierto)
             (heuristica ?h2&:(< ?h2 ?h1))))
  =>
  (modify ?nodo (clase cerrado)))
```

- **Módulo de restricciones**

```
(defmodule RESTRICCIONES
  (import MAIN deftemplate nodo))

(defrule RESTRICCIONES::repeticion-de-nodo
  (declare (auto-focus TRUE))
  (nodo (estado $?actual)
        (camino $?movimientos-1))
  ?nodo <- (nodo (estado $?actual)
                 (camino $?movimientos-1 ? $?))
  =>
  (retract ?nodo))
```

El problema de las fichas con heurística

- **Módulo de reconocimiento y escritura de solución**

```
(defmodule SOLUCION
  (import MAIN deftemplate nodo))

(defrule SOLUCION::reconoce-solucion
  (declare (auto-focus TRUE))
  ?nodo <- (nodo (estado V V V H B B B)
                 (camino $?estados))
  =>
  (retract ?nodo)
  (assert (solucion $?estados)))

(defrule SOLUCION::escribe-solucion
  (solucion $?estados)
  =>
  (bind ?longitud (length $?estados))
  (printout t "La solucion, de longitud " ?longitud
              " es:" crlf)
  (loop-for-count (?i 1 ?longitud)
    (bind ?estado (nth ?i $?estados))
    (printout t ?estado crlf))
  (retract *))
```

El problema de las fichas con heurística

- **Sesión con estadística**

```
CLIPS> (watch statistics)
CLIPS> (reset)
CLIPS> (run)
La solucion, de longitud 15 es:
B B B H V V V
B B B V V V H
B B B V V H V
B B H V V B V
B B V V H B V
B H V V B B V
B V V H B B V
B V V V B B H
B V V V B H B
B V V V H B B
B V V H V B B
H V V B V B B
V V H B V B B
V V V B H B B
V V V H B B B
93 rules fired        Run time is 1.83 seconds.
50.7272727270043 rules per second.
26 mean number of facts (49 maximum).
1 mean number of instances (1 maximum).
5 mean number of activations (12 maximum).
```

Fichas con heurística y coste

- **Coste de un nodo = número de movimientos**
- **Módulos principal**

```
(defmodule MAIN
  (export deftemplate nodo))
(deftemplate MAIN::nodo
  (multislot estado)
  (multislot camino)
  (slot heuristica)
  (slot coste)
  (slot clase (default abierto)))
(defglobal MAIN
  ?*estado-inicial* = (create$ B B B H V V V))

(deffunction heuristica ($?estado)
  (bind ?resultado 0)
  (bind ?verdes-restantes 3)
  (loop-for-count (?i 1 7)
    (bind ?ficha (nth ?i $?estado))
    (if (eq ?ficha B)
        then (bind ?resultado
                   (+ ?resultado ?verdes-restantes))
        else (if (eq ?ficha V)
                 then (bind ?verdes-restantes
                            (- ?verdes-restantes 1)))))
  ?resultado)
```

Fichas con heurística y coste

```
(defrule MAIN::inicial
  =>
  (assert (nodo (estado ?*estado-inicial*)
                (camino (implode$ ?*estado-inicial*))
                (heuristica (heuristica ?*estado-inicial*))
                (coste 0)
                (clase cerrado))))

(defrule MAIN::movimiento-izquierda
  (nodo (estado $?x
                H
                $?y&:(<= (length $?y) 2)
                ?ficha
                $?z)
        (camino $?movimientos)
        (coste ?coste)
        (clase cerrado))
  =>
  (bind $?nuevo-estado (create$ $?x ?ficha $?y H $?z))
  (assert (nodo (estado $?nuevo-estado)
                (camino $?movimientos
                        (implode$ $?nuevo-estado))
                (heuristica (heuristica $?nuevo-estado))
                (coste (+ ?coste 1)))))
```

Fichas con heurística y coste

```
(defrule MAIN::movimiento-derecha
  (nodo (estado $?x
                ?ficha
                $?y&:(<= (length $?y) 2)
                H
                $?z)
        (camino $?movimientos)
        (coste ?coste)
        (clase cerrado))
  =>
  (bind $?nuevo-estado (create$ $?x H $?y ?ficha $?z))
  (assert (nodo (estado $?nuevo-estado)
                (camino $?movimientos
                        (implode$ $?nuevo-estado))
                (heuristica (heuristica $?nuevo-estado))
                (coste (+ ?coste 1)))))
```

Fichas con heurística y coste

```
(defrule MAIN::pasa-el-mejor-a-cerrado
  (declare (salience -10))
  ?nodo <- (nodo (clase abierto)
                 (heuristica ?h1)
                 (coste ?c1))
  (not (nodo (clase abierto)
             (heuristica ?h2&:(< ?h2 ?h1))))
  (not (nodo (clase abierto) (heuristica ?h1)
             (coste ?c2&:(< ?c2 ?c1))))
  =>
  (modify ?nodo (clase cerrado)))
```

• Módulo de restricciones

```
(defmodule RESTRICCIONES
  (import MAIN deftemplate nodo))

(defrule RESTRICCIONES::repeticion-de-nodo
  (declare (auto-focus TRUE))
  (nodo (estado $?actual)
        (coste ?coste-1))
  ?nodo <- (nodo (estado $?actual)
                 (coste ?coste-2&:(> ?coste-2 ?coste-1)))
  =>
  (retract ?nodo))
```

Fichas con heurística y coste

- **Módulo de reconocimiento y escritura de solución**

```
(defmodule SOLUCION
  (import MAIN deftemplate nodo))

(defrule SOLUCION::reconoce-solucion
  (declare (auto-focus TRUE))
  ?nodo <- (nodo (estado V V V H B B B)
                 (camino $?estados))
  =>
  (retract ?nodo)
  (assert (solucion $?estados)))

(defrule SOLUCION::escribe-solucion
  (solucion $?estados)
  =>
  (bind ?longitud (length $?estados))
  (printout t "La solucion, de longitud " ?longitud
              " es:" crlf)
  (loop-for-count (?i 1 ?longitud)
    (bind ?estado (nth ?i $?estados))
    (printout t ?estado crlf))
  (retract *))
```

Fichas con heurística y coste

• Sesión con estadística

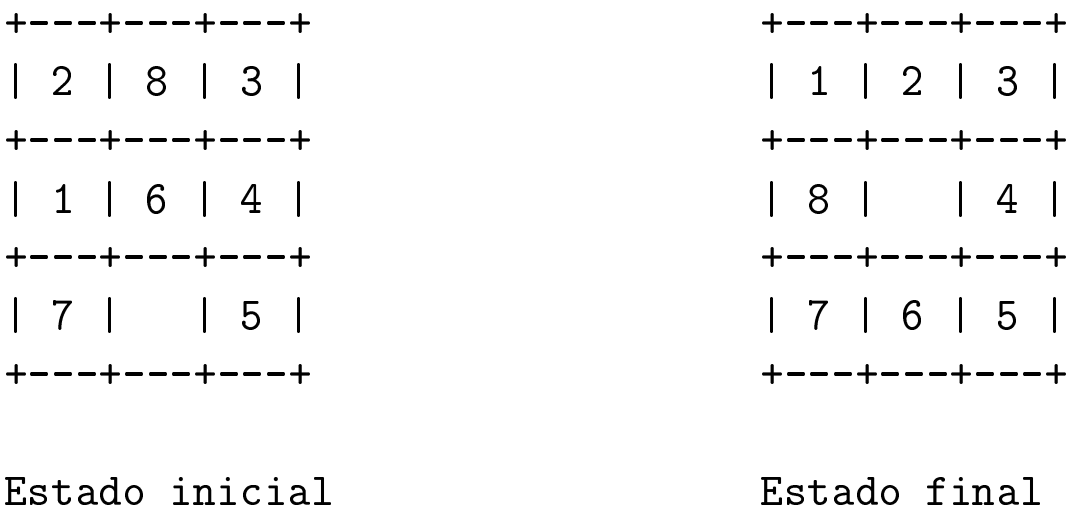
```
CLIPS> (load "fichas-3.clp")
+%:!*********
TRUE
CLIPS> (watch statistics)
CLIPS> (reset)
CLIPS> (run)
La solucion, de longitud 13 es:
B B B H V V V
B B B V V H V
B B H V V B V
B B V V H B V
B H V V B B V
B V V H B B V
H V V B B B V
V V H B B B V
V V B H B B V
V V B V B B H
V V B V B H B
V V H V B B B
V V V H B B B
120 rules fired        Run time is 0.65 seconds.
184.6153846170378 rules per second.
25 mean number of facts (47 maximum).
1 mean number of instances (1 maximum).
5 mean number of activations (11 maximum).
```

Comparación de soluciones

	Sol 1	Sol 2 heur.	Sol 3 heur. coste
Longitud de la solución	83	15	13
Tiempo (segundos)	34.50	1.83	0.65
Número de disparos	238	93	120
Número máximo de hechos	84	49	47
Número medio de hechos	43	26	25
Número máximo de activaciones	205	12	11
Número medio de activaciones	103	5	5

Problema del 8-puzzle

• Enunciado



• Módulo principal

```
(defmodule MAIN
  (export deftemplate nodo))

(deftemplate MAIN::nodo
  (multislot estado)
  (multislot camino))

(deffacts MAIN::nodo-inicial
  (nodo (estado 2 8 3 1 6 4 7 H 5)
        (camino)))
```

Problema del 8-puzzle

```
(defrule MAIN::arriba
  (nodo (estado $?a ?b ?c ?d H $?e)
        (camino $?movimientos))
  =>
  (assert (nodo (estado $?a H ?c ?d ?b $?e)
                (camino $?movimientos ^))))

(defrule MAIN::abajo
  (nodo (estado $?a H ?b ?c ?d $?e)
        (camino $?movimientos))
  =>
  (assert (nodo (estado $?a ?d ?b ?c H $?e)
                (camino $?movimientos v))))

(defrule MAIN::izquierda
  (nodo (estado $?a&:(neq (mod (length $?a) 3) 2)
                ?b H $?c)
        (camino $?movimientos))
  =>
  (assert (nodo (estado $?a H ?b $?c)
                (camino $?movimientos <))))
```

Problema del 8-puzzle

```
(defrule MAIN::derecha
  (nodo (estado $?a H ?b
                $?c&:(neq (mod (length $?c) 3) 2))
        (camino $?movimientos))
  =>
  (assert (nodo (estado $?a ?b H $?c)
                (camino $?movimientos >))))
```

• Módulo de restricciones

```
(defmodule RESTRICCIONES
  (import MAIN deftemplate nodo))

(defrule RESTRICCIONES::repeticion-de-nodo
  (declare (auto-focus TRUE))
  (nodo (estado $?actual)
        (camino $?movimientos-1))
  ?nodo <- (nodo (estado $?actual)
                 (camino $?movimientos-1 ? $?))
  =>
  (retract ?nodo))
```

Problema del 8-puzzle

- **Módulo solución**

```
(defmodule SOLUCION
  (import MAIN deftemplate nodo))

(defrule SOLUCION::reconoce-solucion
  (declare (auto-focus TRUE))
  ?nodo <- (nodo (estado 1 2 3 8 H 4 7 6 5)
                 (camino $?movimientos))
  =>
  (retract ?nodo)
  (assert (solucion $?movimientos)))

(defrule SOLUCION::escribe-solucion
  (solucion $?movimientos)
  =>
  (printout t "Solucion: " $?movimientos crlf)
  (halt))
```

Problema del 8-puzzle

- **Sesión**

```
CLIPS> (reset)
CLIPS> (run)
[PRCCODE4] Execution halted
637 rules fired        Run time is 2688.38 seconds.
0.2369453760934392 rules per second.
154 mean number of facts (306 maximum).
1 mean number of instances (1 maximum).
197 mean number of activations (389 maximum).

CLIPS> (set-strategy breadth)
depth
CLIPS> (reset)
CLIPS> (run)
Solucion: (^ ^ < v >)
123 rules fired        Run time is 2.62 seconds.
47.00636942678209 rules per second.
32 mean number of facts (61 maximum).
1 mean number of instances (1 maximum).
39 mean number of activations (73 maximum).
```

Problema del 8-puzzle (con heurística)

- **Heurística**
  - **Definición: número de piezas descolocadas**
  - **Heurística del estado inicial: 5**

- **Módulo principal**

```
(defmodule MAIN
  (export deftemplate nodo))

(deftemplate MAIN::nodo
  (multislot estado)
  (multislot camino)
  (slot heuristica)
  (slot clase (default abierto)))
```

Problema del 8-puzzle (con heurística)

```
(defglobal MAIN
  ?*estado-inicial* = (create$ 2 8 3 1 6 4 7 H 5)
  ?*estado-final*   = (create$ 1 2 3 8 H 4 7 6 5))

(deffunction MAIN::heuristica ($?estado)
  (bind ?res 0)
  (loop-for-count (?i 1 9)
    (if (neq (nth ?i $?estado)
             (nth ?i ?*estado-final*))
        then (bind ?res (+ ?res 1))))
  ?res)

(defrule MAIN::inicial
  =>
  (assert (nodo (estado ?*estado-inicial*)
                (camino)
                (heuristica (heuristica ?*estado-inicial*))
                (clase cerrado))))
```

Problema del 8-puzzle (con heurística)

```
(defrule MAIN::arriba
  (nodo (estado $?a ?b ?c ?d H $?e)
        (camino $?movimientos)
        (clase cerrado))
  =>
  (bind $?nuevo-estado (create$ $?a H ?c ?d ?b $?e))
  (assert (nodo (estado $?nuevo-estado)
                (camino $?movimientos ^)
                (heuristica (heuristica $?nuevo-estado)))))

(defrule MAIN::abajo
  (nodo (estado $?a H ?b ?c ?d $?e)
        (camino $?movimientos)
        (clase cerrado))
  =>
  (bind $?nuevo-estado (create$ $?a ?d ?b ?c H $?e))
  (assert (nodo (estado $?nuevo-estado)
                (camino $?movimientos v)
                (heuristica (heuristica $?nuevo-estado)))))
```

Problema del 8-puzzle (con heurística)

```
(defrule MAIN::izquierda
  (nodo (estado $?a&:(neq (mod (length $?a) 3) 2)
                 ?b H $?c)
        (camino $?movimientos)
        (clase cerrado))
  =>
  (bind $?nuevo-estado (create$ $?a H ?b $?c))
  (assert (nodo (estado $?nuevo-estado)
                (camino $?movimientos <)
                (heuristica (heuristica $?nuevo-estado)))))

(defrule MAIN::derecha
  (nodo (estado $?a H ?b
                $?c&:(neq (mod (length $?c) 3) 2))
        (camino $?movimientos)
        (clase cerrado))
  =>
  (bind $?nuevo-estado (create$ $?a ?b H $?c))
  (assert (nodo (estado $?nuevo-estado)
                (camino $?movimientos >)
                (heuristica (heuristica $?nuevo-estado)))))
```

Problema del 8-puzzle (con heurística)

```
(defrule MAIN::pasa-el-mejor-a-cerrado
  (declare (salience -10))
  ?nodo <- (nodo (clase abierto)
                 (heuristica ?h1))
  (not (nodo (clase abierto)
             (heuristica ?h2&:(< ?h2 ?h1))))
  =>
  (modify ?nodo (clase cerrado)))
```

• Módulo de restricciones

```
(defmodule RESTRICCIONES
  (import MAIN deftemplate nodo))

(defrule RESTRICCIONES::repeticion-de-nodo
  (declare (auto-focus TRUE))
  (nodo (estado $?actual)
        (camino $?movimientos-1))
  ?nodo <- (nodo (estado $?actual)
                 (camino $?movimientos-1 ? $?))
  =>
  (retract ?nodo))
```

Problema del 8-puzzle (con heurística)

- **Módulo solución**

```
(defmodule SOLUCION
  (import MAIN deftemplate nodo))

(defrule SOLUCION::reconoce-solucion
  (declare (auto-focus TRUE))
  ?nodo <- (nodo (heuristica 0)
                 (camino $?movimientos))
  =>
  (retract ?nodo)
  (assert (solucion $?movimientos)))

(defrule SOLUCION::escribe-solucion
  (solucion $?movimientos)
  =>
  (printout t "Solucion: " $?movimientos crlf)
  (halt))
```

Problema del 8-puzzle (con heurística)

- **Sesión**

```
CLIPS> (reset)
CLIPS> (watch statistics)
CLIPS> (run)
Solucion: (^ ^ < v >)
31 rules fired        Run time is 0.33 seconds.
92.99999999995771 rules per second.
10 mean number of facts (15 maximum).
1 mean number of instances (1 maximum).
3 mean number of activations (6 maximum).
```

Comparación de soluciones

	Sol 1	Sol 2 heurist.
Longitud de la solucion	6	6
Tiempo (segundos)	2.62	0.33
Número de disparos	123	31
Número máximo de hechos	61	15
Número medio de hechos	31	10
Número máximo de activaciones	73	6
Número medio de activaciones	39	3

Problema del 8-puzzle (con heurística)

- **Sesión con traza reducida**

```
CLIPS> (watch facts)
CLIPS> (watch rules)
CLIPS> (watch activations)
CLIPS> (reset)
==> Focus MAIN
==> f-0     (initial-fact)
==> Activation 0      inicial: f-0
CLIPS> (run)
FIRE    1 inicial: f-0
==> f-1     (nodo (est 2 8 3 1 6 4 7 H 5) (heu 5) (cla cer))
==> Activation 0      derecha: f-1
==> Activation 0      izquierda: f-1
==> Activation 0      arriba: f-1
FIRE    2 arriba: f-1
==> f-2     (nodo (est 2 8 3 1 H 4 7 6 5) (heu 3) (cla abi))
==> Activation -10    pasa-el-mejor-a-cerrado: f-2,
FIRE    3 izquierda: f-1
==> f-3     (nodo (est 2 8 3 1 6 4 H 7 5) (heu 6) (cla abi))
FIRE    4 derecha: f-1
==> f-4     (nodo (est 2 8 3 1 6 4 7 5 H) (heu 6) (cla abi))
```

Problema del 8-puzzle (con heurística)

```
FIRE    5 pasa-el-mejor-a-cerrado: f-2,
<== f-2     (nodo (est 2 8 3 1 H 4 7 6 5) (heu 3) (cla abi))
==> Activation -10    pasa-el-mejor-a-cerrado: f-4,
==> Activation -10    pasa-el-mejor-a-cerrado: f-3,
==> f-5     (nodo (est 2 8 3 1 H 4 7 6 5) (heu 3) (cla cer))
==> Activation 0      derecha: f-5
==> Activation 0      izquierda: f-5
==> Activation 0      abajo: f-5
==> Activation 0      arriba: f-5
FIRE    6 arriba: f-5
==> f-6     (nodo (est 2 H 3 1 8 4 7 6 5) (heu 4) (cla abi))
<== Activation -10    pasa-el-mejor-a-cerrado: f-4,
<== Activation -10    pasa-el-mejor-a-cerrado: f-3,
==> Activation -10    pasa-el-mejor-a-cerrado: f-6,
FIRE    7 abajo: f-5
==> f-7     (nodo (est 2 8 3 1 6 4 7 H 5) (heu 5) (cla abi))
==> Focus RESTRICCIONES from MAIN
==> Activation 0      repeticion-de-nodo: f-1,f-7
FIRE    8 repeticion-de-nodo: f-1,f-7
<== f-7     (nodo (est 2 8 3 1 6 4 7 H 5) (heu 5) (cla abi))
<== Focus RESTRICCIONES to MAIN
FIRE    9 izquierda: f-5
==> f-8     (nodo (est 2 8 3 H 1 4 7 6 5) (heu 4) (cla abi))
==> Activation -10    pasa-el-mejor-a-cerrado: f-8,
```

Problema del 8-puzzle (con heurística)

```
FIRE   10 derecha: f-5
==> f-9     (nodo (est 2 8 3 1 4 H 7 6 5) (heu 5) (cla abi))
FIRE   11 pasa-el-mejor-a-cerrado: f-8,
<== f-8     (nodo (est 2 8 3 H 1 4 7 6 5) (heu 4) (cla abi))
==> f-10    (nodo (est 2 8 3 H 1 4 7 6 5) (heu 4) (cla cer))
==> Activation 0      derecha: f-10
==> Activation 0      abajo: f-10
==> Activation 0      arriba: f-10
FIRE   12 arriba: f-10
==> f-11    (nodo (est H 8 3 2 1 4 7 6 5) (heu 4) (cla abi))
==> Activation -10    pasa-el-mejor-a-cerrado: f-11,
FIRE   13 abajo: f-10
==> f-12    (nodo (est 2 8 3 7 1 4 H 6 5) (heu 5) (cla abi))
FIRE   14 derecha: f-10
==> f-13    (nodo (est 2 8 3 1 H 4 7 6 5) (heu 3) (cla abi))
==> Focus RESTRICCIONES from MAIN
==> Activation 0      repeticion-de-nodo: f-5,f-13
<== Activation -10    pasa-el-mejor-a-cerrado: f-11,
<== Activation -10    pasa-el-mejor-a-cerrado: f-6,
==> Activation -10    pasa-el-mejor-a-cerrado: f-13,
FIRE   15 repeticion-de-nodo: f-5,f-13
<== f-13    (nodo (est 2 8 3 1 H 4 7 6 5) (heu 3) (cla abi))
<== Activation -10    pasa-el-mejor-a-cerrado: f-13,
==> Activation -10    pasa-el-mejor-a-cerrado: f-11,
==> Activation -10    pasa-el-mejor-a-cerrado: f-6,
<== Focus RESTRICCIONES to MAIN
```

Problema del 8-puzzle (con heurística)

```
FIRE   16 pasa-el-mejor-a-cerrado: f-6,
<== f-6     (nodo (est 2 H 3 1 8 4 7 6 5) (heu 4) (cla abi))
==> f-14    (nodo (est 2 H 3 1 8 4 7 6 5) (heu 4) (cla cer))
==> Activation 0      derecha: f-14
==> Activation 0      izquierda: f-14
==> Activation 0      abajo: f-14
FIRE   17 abajo: f-14
==> f-15    (nodo (est 2 8 3 1 H 4 7 6 5) (heu 3) (cla abi))
==> Focus RESTRICCIONES from MAIN
==> Activation 0      repeticion-de-nodo: f-5,f-15
<== Activation -10    pasa-el-mejor-a-cerrado: f-11,
==> Activation -10    pasa-el-mejor-a-cerrado: f-15,
FIRE   18 repeticion-de-nodo: f-5,f-15
<== f-15    (nodo (est 2 8 3 1 H 4 7 6 5) (heu 3) (cla abi))
<== Activation -10    pasa-el-mejor-a-cerrado: f-15,
==> Activation -10    pasa-el-mejor-a-cerrado: f-11,
<== Focus RESTRICCIONES to MAIN
FIRE   19 izquierda: f-14
==> f-16    (nodo (est H 2 3 1 8 4 7 6 5) (heu 3) (cla abi))
<== Activation -10    pasa-el-mejor-a-cerrado: f-11,
==> Activation -10    pasa-el-mejor-a-cerrado: f-16,
FIRE   20 derecha: f-14
==> f-17    (nodo (est 2 3 H 1 8 4 7 6 5) (heu 5) (cla abi))
```

Problema del 8-puzzle (con heurística)

```
FIRE   21 pasa-el-mejor-a-cerrado: f-16,
<== f-16    (nodo (est H 2 3 1 8 4 7 6 5) (heu 3) (cla abi))
==> Activation -10    pasa-el-mejor-a-cerrado: f-11,
==> f-18    (nodo (est H 2 3 1 8 4 7 6 5) (heu 3) (cla cer))
==> Activation 0      derecha: f-18
==> Activation 0      abajo: f-18
FIRE   22 abajo: f-18
==> f-19    (nodo (est 1 2 3 H 8 4 7 6 5) (heu 2) (cla abi))
<== Activation -10    pasa-el-mejor-a-cerrado: f-11,
==> Activation -10    pasa-el-mejor-a-cerrado: f-19,
FIRE   23 derecha: f-18
==> f-20    (nodo (est 2 H 3 1 8 4 7 6 5) (heu 4) (cla abi))
==> Focus RESTRICCIONES from MAIN
==> Activation 0      repeticion-de-nodo: f-14,f-20
FIRE   24 repeticion-de-nodo: f-14,f-20
<== f-20    (nodo (est 2 H 3 1 8 4 7 6 5) (heu 4) (cla abi))
<== Focus RESTRICCIONES to MAIN
FIRE   25 pasa-el-mejor-a-cerrado: f-19,
<== f-19    (nodo (est 1 2 3 H 8 4 7 6 5) (heu 2) (cla abi))
==> Activation -10    pasa-el-mejor-a-cerrado: f-11,
==> f-21    (nodo (est 1 2 3 H 8 4 7 6 5) (heu 2) (cla cer))
==> Activation 0      derecha: f-21
==> Activation 0      abajo: f-21
==> Activation 0      arriba: f-21
```

Problema del 8-puzzle (con heurística)

```
FIRE   26 arriba: f-21
==> f-22    (nodo (est H 2 3 1 8 4 7 6 5) (heu 3) (cla abi))
==> Focus RESTRICCIONES from MAIN
==> Activation 0      repeticion-de-nodo: f-18,f-22
<== Activation -10    pasa-el-mejor-a-cerrado: f-11,
==> Activation -10    pasa-el-mejor-a-cerrado: f-22,
FIRE   27 repeticion-de-nodo: f-18,f-22
<== f-22    (nodo (est H 2 3 1 8 4 7 6 5) (heu 3) (cla abi))
<== Activation -10    pasa-el-mejor-a-cerrado: f-22,
==> Activation -10    pasa-el-mejor-a-cerrado: f-11,
<== Focus RESTRICCIONES to MAIN
FIRE   28 abajo: f-21
==> f-23    (nodo (est 1 2 3 7 8 4 H 6 5) (heu 3) (cla abi))
<== Activation -10    pasa-el-mejor-a-cerrado: f-11,
==> Activation -10    pasa-el-mejor-a-cerrado: f-23,
FIRE   29 derecha: f-21
==> f-24    (nodo (est 1 2 3 8 H 4 7 6 5) (heu 0) (cla abi))
==> Focus SOLUCION from MAIN
==> Activation 0      reconoce-solucion: f-24
<== Activation -10    pasa-el-mejor-a-cerrado: f-23,
==> Activation -10    pasa-el-mejor-a-cerrado: f-24,
```

Problema del 8-puzzle (con heurística)

```
FIRE   30 reconoce-solucion: f-24
<== f-24    (nodo (est 1 2 3 8 H 4 7 6 5) (heu 0) (cla abi))
<== Activation -10    pasa-el-mejor-a-cerrado: f-24,
==> Activation -10    pasa-el-mejor-a-cerrado: f-23,
==> f-25    (solucion ^ ^ < v >)
==> Activation 0      escribe-solucion: f-25
FIRE   31 escribe-solucion: f-25
Solucion: (^ ^ < v >)
```