UNIVERSITAT POLITÈCNICA DE VALÈNCIA

# Introducción a los sistemas basados en reglas con CLIPS [1]

Alfons Juan
Albert Sanchis
Jorge Civera

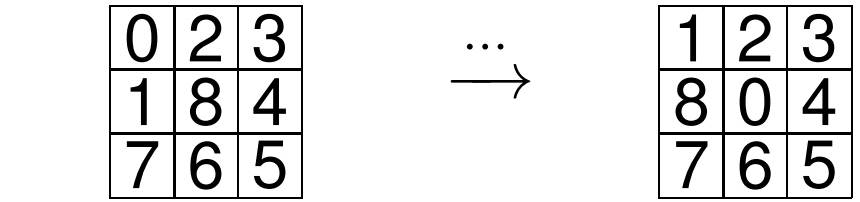
DSIC
Departamento de Sistemas Informáticos y Computación

[1] Para una correcta visualización, se requiere Acrobat Reader v. 7.0 o superior

# Índice

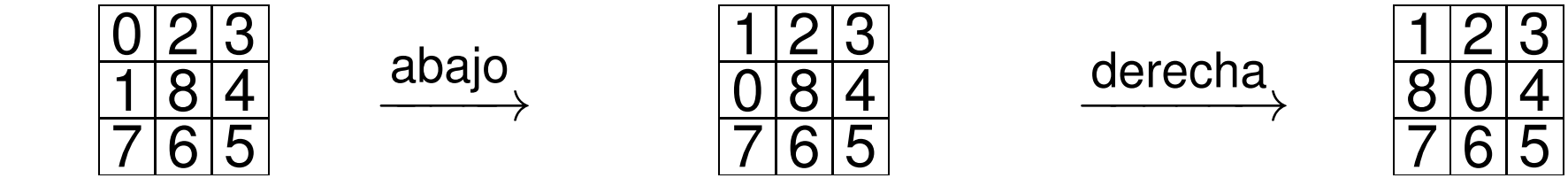
# 1. Problema: el 8-puzzle

Dado un hecho inicial, debemos llegar a un estado objetivo indicado abajo con los movimientos del hueco (ficha 0): derecha, izquierda, abajo y arriba.




Estado inicial Objetivo

Solución con el estado inicial anterior:




Estado inicial Estado intermedio Estado final

# 2. Sistemas basados en reglas con CLIPS

***CLIPS*** es una herramienta para construir SBRs con tres componentes:

1. ***Base de hechos (BH):***
   - Cada estado del problema suele representarse con un único hecho de acuerdo a un cierto patrón de hecho que definimos.
   - A cada paso de ejecución, los hechos de la BH representan estados del problema ya explorados o por explorar.
   - El resto de hechos son información *estática* del problema.

2. ***Base de reglas (BR):***
   - Cada posible acción aplicable a uno o más estados del problema suele representarse con una única regla `izq =>dcha`.
   - La parte izquierda elige el conjunto de estados aplicable.
   - La parte derecha suele añadir nuevos hechos a la BH.

3. ***Motor de inferencia:*** instanciación, selección y ejecución de reglas..

# Base de hechos y base de reglas

La base de hechos se define como:

```
(deffacts <nombre> [<comentario>] <hecho>*)
```

donde el formato de cada hecho es:

```
(<símbolo> <const>*)
```

El formato de las reglas es:

```
(defrule <nombre> <condiciones>* => <órdenes>*)
```

donde las condiciones pueden ser ***patrones*** o ***tests:***

```
(<símbolo> [<const>|?<var>|$?<var>]*)
(test (<función> <expresión>*))
```

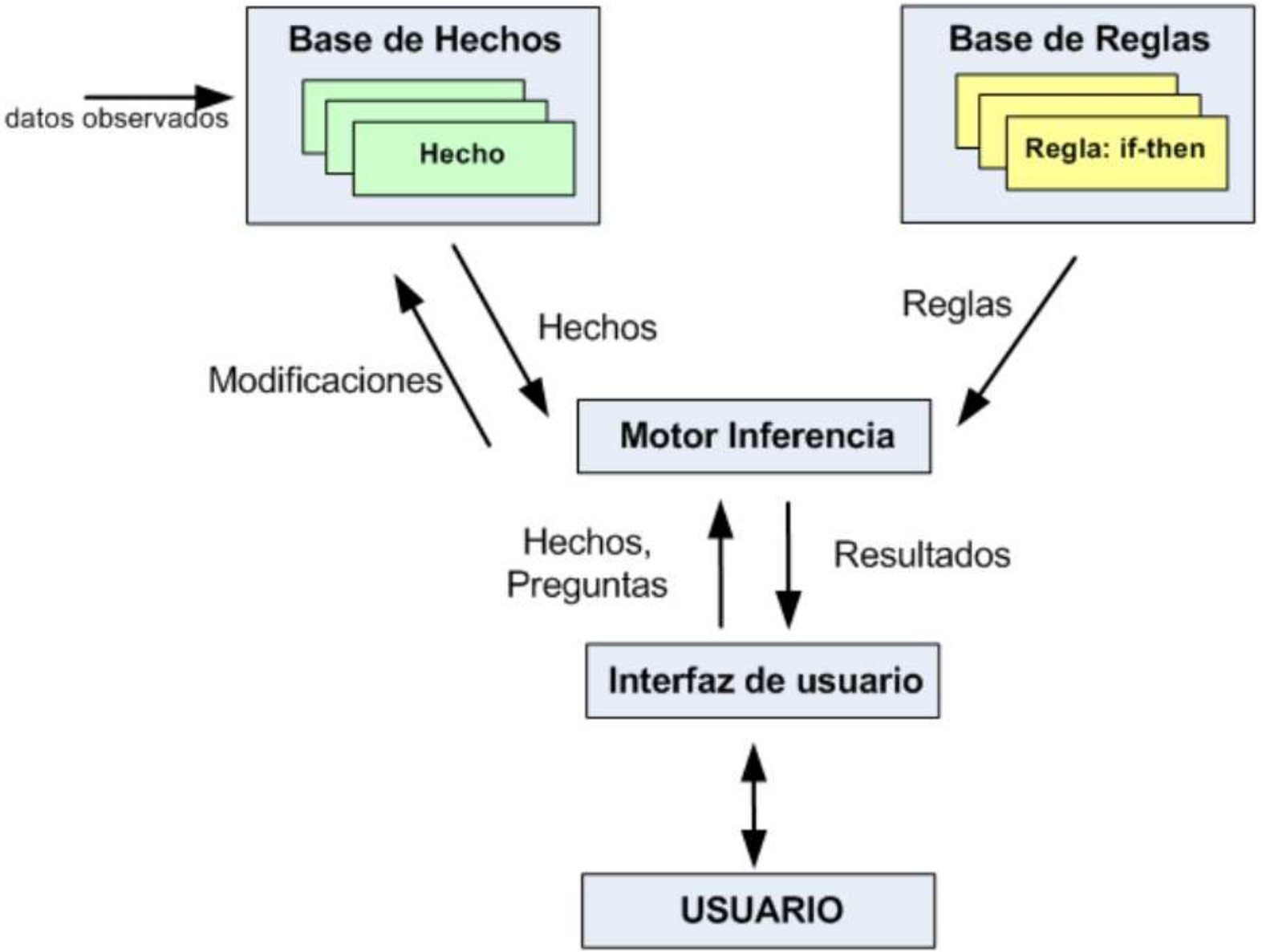
y las órdenes permiten insertar o borrar hechos, etc.:

```
(assert <hecho>+)
(retract <índice-hecho>+)
(printout "¡Eureka!" crlf)
(halt)
```

# SBR sencillo para el 8-puzzle

```
(deffacts bhini (puzzle 0 2 3 1 8 4 7 6 5))
(defrule izquierda
  (puzzle $?x ?y 0 $?z)
  (test (<> (length$ $?x) 2))
  (test (<> (length$ $?x) 5)) =>
  (assert (puzzle $?x 0 ?y $?z)))
(defrule derecha
  (puzzle $?x 0 ?y $?z)
  (test (<> (length$ $?x) 2))
  (test (<> (length$ $?x) 5)) =>
  (assert (puzzle $?x ?y 0 $?z)))
(defrule arriba
  (puzzle $?x ?a ?b ?c 0 $?y) =>
  (assert (puzzle $?x 0 ?b ?c ?a $?y)))
(defrule abajo
  (puzzle $?x 0 ?a ?b ?c $?z) =>
  (assert (puzzle $?x ?c ?a ?b 0 $?z)))
(defrule objetivo
  (puzzle 1 2 3 8 0 4 7 6 5) =>
  (printout t "¡Solución encontrada!" crlf)
  (halt))
```

# Sistemas basados en reglas con CLIPS



# 3. Motor de inferencia

- ***Entrada:*** base de hechos y base de reglas iniciales, $BH$ y $BR$
- ***Salida:*** base de hechos final, $BH$
- ***Método:***

  $CC = \emptyset$ // conjunto conflicto de instancias de reglas

  **repetir**

    //añadimos nuevas instancias a $CC$ usando nuevos hechos:
    $CC =$ Instancia$(BH, BR, CC)$

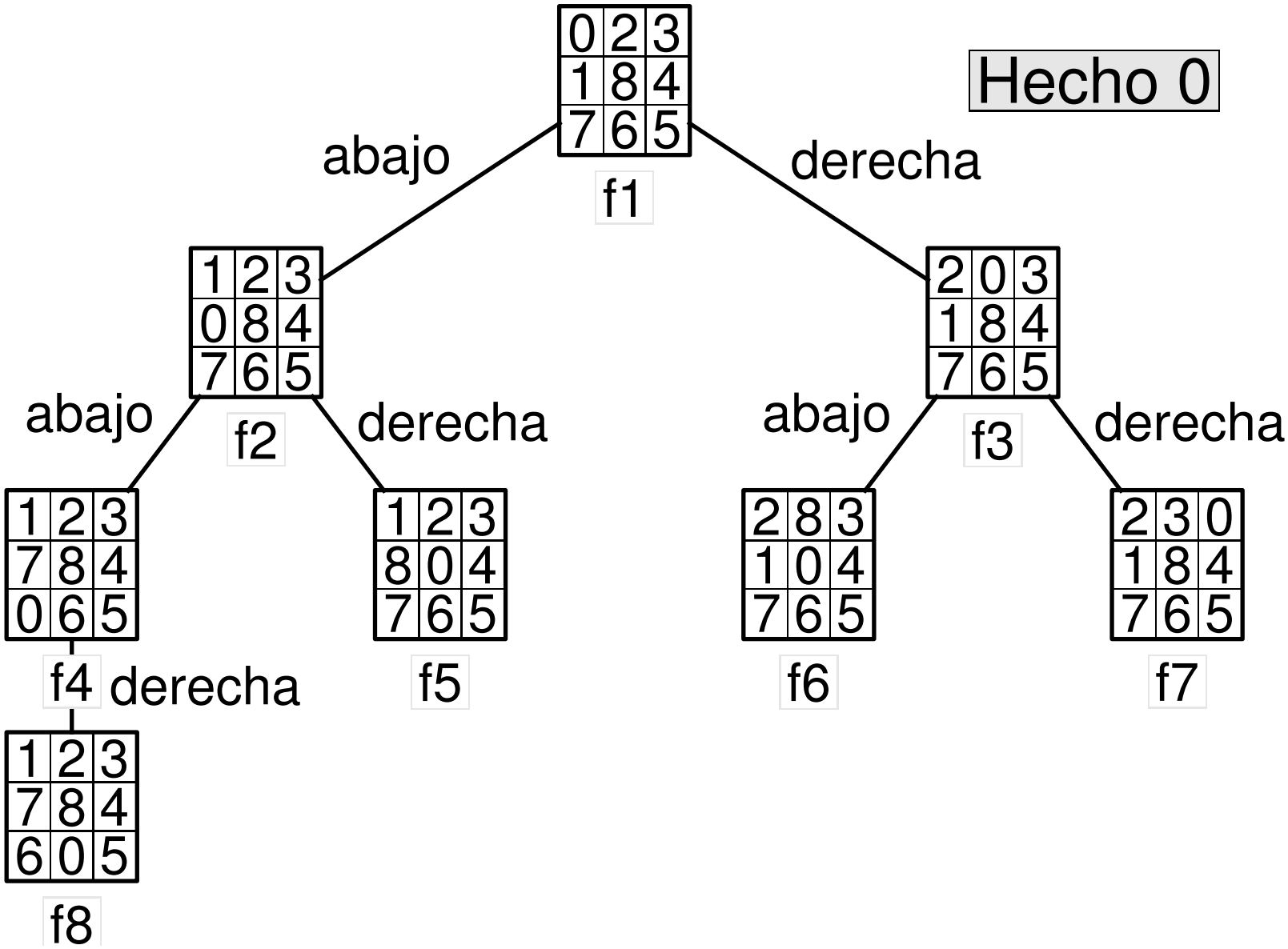
    **si** $CC = \emptyset$: **break** // objetivo no conseguido

    // seleccionamos una instancia con algún criterio:
    $InstRule =$ Selecciona$(CC)$

    // ejecutamos $InstRule$ y actualizamos $BH$ y $CC$:
    $(BH, CC) =$ Ejecuta$(BH, CC, InstRule)$

  **hasta** objetivo conseguido

# Traza en anchura representada con un árbol



# Motor de inferencia

- No duplicidad de hechos en $BH$ (opcional en CLIPS)
- ***Refracción:***

  Una regla sólo se puede instanciar una vez con el mismo hecho y con los mismos valores de variables
- No duplicidad y refracción evitan activación infinita de reglas
- Un nuevo hecho posibilita que cualquier regla pueda activarse
- ***Estrategias de resolución de conflictos:***

  Anchura, profundidad, aleatoria, priorizada

```
(deffacts bhini (puzzle 0 2 3 1 8 4 7 6 5))
(defrule izquierda
(puzzle $?x ?y 0 $?z)
(test (<> (length$ $?x) 2))
(test (<> (length$ $?x) 5)) =>
(assert (puzzle $?x 0 ?y $?z)))
(defrule derecha
(puzzle $?x 0 ?y $?z)
(test (<> (length$ $?x) 2))
(test (<> (length$ $?x) 5)) =>
(assert (puzzle $?x ?y 0 $?z)))
(defrule arriba
(puzzle $?x ?a ?b ?c 0 $?y) =>
(assert (puzzle $?x 0 ?b ?c ?a $?y)))
(defrule abajo
(puzzle $?x 0 ?a ?b ?c $?z) =>
(assert (puzzle $?x ?c ?a ?b 0 $?z)))
(defrule objetivo
(puzzle 1 2 3 8 0 4 7 6 5) =>
(printout t "¡Solución encontrada!" crlf)
(halt))
(set-strategy breadth)
(watch facts)
(watch activations)
(reset)
(run)
(exit)
```

```
CLIPS (V6.24 06/15/06)
==> f-0     (initial-fact)
==> f-1     (puzzle 0 2 3 1 8 4 7 6 5)
==> Activation 0      abajo: f-1
==> Activation 0      derecha: f-1
CLIPS> (run)
==> f-2     (puzzle 1 2 3 0 8 4 7 6 5)
==> Activation 0      abajo: f-2
==> Activation 0      arriba: f-2
==> Activation 0      derecha: f-2
==> f-3     (puzzle 2 0 3 1 8 4 7 6 5)
==> Activation 0      abajo: f-3
==> Activation 0      derecha: f-3
==> Activation 0      izquierda: f-3
==> f-4     (puzzle 1 2 3 7 8 4 0 6 5)
==> Activation 0      arriba: f-4
==> Activation 0      derecha: f-4
==> f-5     (puzzle 1 2 3 8 0 4 7 6 5)
==> Activation 0      objetivo: f-5
==> Activation 0      abajo: f-5
==> Activation 0      arriba: f-5
==> Activation 0      derecha: f-5
==> Activation 0      izquierda: f-5
==> f-6     (puzzle 2 8 3 1 0 4 7 6 5)
==> Activation 0      abajo: f-6
==> Activation 0      arriba: f-6
==> Activation 0      derecha: f-6
==> Activation 0      izquierda: f-6
==> f-7     (puzzle 2 3 0 1 8 4 7 6 5)
==> Activation 0      abajo: f-7
==> Activation 0      izquierda: f-7
==> f-8     (puzzle 1 2 3 7 8 4 6 0 5)
==> Activation 0      arriba: f-8
==> Activation 0      derecha: f-8
==> Activation 0      izquierda: f-8
¡Solución encontrada!
```

# Ejercicio Práctica 1

```
(deffacts datos (lista 4 5 3 46 12 10))
(defrule ordenar
  ?f <- (lista $?x ?y ?z $?w)
  (test (< ?z ?y))
  =>
  (retract ?f)
  (assert (lista $?x ?z ?y $?w)))
(set-strategy breadth)
(watch facts)
(watch activations)
(reset)
(run)
(exit)
```

Realiza una traza mostrando la base de hechos y conjunto conflicto.

```
CLIPS (V6.24 06/15/06)
CLIPS> (reset)
==> f-0     (initial-fact)
==> f-1     (lista 4 5 3 46 12 10)
==> Activation 0      ordenar: f-1
==> Activation 0      ordenar: f-1
==> Activation 0      ordenar: f-1
CLIPS> (run)
<== f-1     (lista 4 5 3 46 12 10)
<== Activation 0      ordenar: f-1
<== Activation 0      ordenar: f-1
==> f-2     (lista 4 5 3 46 10 12)
==> Activation 0      ordenar: f-2
==> Activation 0      ordenar: f-2
<== f-2     (lista 4 5 3 46 10 12)
<== Activation 0      ordenar: f-2
==> f-3     (lista 4 5 3 10 46 12)
==> Activation 0      ordenar: f-3
==> Activation 0      ordenar: f-3
<== f-3     (lista 4 5 3 10 46 12)
<== Activation 0      ordenar: f-3
==> f-4     (lista 4 5 3 10 12 46)
==> Activation 0      ordenar: f-4
<== f-4     (lista 4 5 3 10 12 46)
==> f-5     (lista 4 3 5 10 12 46)
==> Activation 0      ordenar: f-5
<== f-5     (lista 4 3 5 10 12 46)
==> f-6     (lista 3 4 5 10 12 46)
```

# Ejercicio (examen 2/11/2015, cuestión 1)

```
(deffacts bh (lista a b a b a))
(defrule R1
  ?f <- (lista ?x $?y ?x $?z) =>
;  (retract ?f)
  (assert (lista $?y ?x $?z))
  (printout t "La lista se ha modificado" crlf))
;(set-strategy breadth)
;(watch facts)
;(watch activations)
;(reset)
;(run)
;(exit)
```

¿Cuántas veces se muestra el mensaje?

```
CLIPS (V6.24 06/15/06)
CLIPS> (reset)
==> f-0     (initial-fact)
==> f-1     (lista a b a b a)
==> Activation 0      R1: f-1
==> Activation 0      R1: f-1
CLIPS> (run)
<== f-1     (lista a b a b a)
<== Activation 0      R1: f-1
==> f-2     (lista b a b a)
==> Activation 0      R1: f-2
La lista se ha modificado
<== f-2     (lista b a b a)
==> f-3     (lista a b a)
==> Activation 0      R1: f-3
La lista se ha modificado
<== f-3     (lista a b a)
==> f-4     (lista b a)
La lista se ha modificado
```

# Ejercicio 4 Tema 2

Sea un SBR cuya BH inicial es BH={(lista 1 2 3 4)} y cuya Base de Reglas se compone de las siguientes dos reglas:

```
(defrule R1
?f <- (lista ?x $?z) =>
(retract ?f)
(assert (lista ?z))
(assert (elemento ?x)))
```

```
(defrule R2
?f <- (elemento ?x)
(elemento ?y)
(test (< ?x ?y)) =>
(retract ?f)
(assert (lista-new ?x ?y))
```

asumiendo una estrategia de búsqueda en anchura:

a) ¿Cuál sería el estado final de la BH? Realiza la traza.

b) ¿Y si la BH inicial fuera BH={(lista 1 2 2 4)}?

c) ¿Y si eliminamos (retract ?f) de R1 con BH={(lista 1 2 3 4)}?

d) ¿Y si eliminamos (retract ?f) de R2 con BH={(lista 1 2 3 4)}?

```
(deffacts bh (lista 1 2 2 4))
(defrule R1
  ?f <- (lista ?x $?z) =>
  (retract ?f)
  (assert (lista $?z))
  (assert (elemento ?x)))
(defrule R2
  ?f <- (elemento ?x)
        (elemento ?y)
  (test (< ?x ?y)) =>
  (retract ?f)
  (assert (lista-new ?x ?y)))
;;(watch facts)
;;(watch activations)
;;(reset)
;;(run)
;;(exit)
```

```
CLIPS (V6.24 06/15/06)
CLIPS> (reset)
==> f-0     (initial-fact)
==> f-1     (lista 1 2 3 4)
==> Activation 0      R1: f-1
CLIPS> (run)
<== f-1     (lista 1 2 3 4)
==> f-2     (lista 2 3 4)
==> Activation 0      R1: f-2
==> f-3     (element 1)
<== f-2     (lista 2 3 4)
==> f-4     (lista 3 4)
==> Activation 0      R1: f-4
==> f-5     (element 2)
==> Activation 0      R2: f-3,f-5
<== f-4     (lista 3 4)
==> f-6     (lista 4)
==> Activation 0      R1: f-6
==> f-7     (element 3)
==> Activation 0      R2: f-5,f-7
==> Activation 0      R2: f-3,f-7
<== f-3     (element 1)
<== Activation 0      R2: f-3,f-7
==> f-8     (lista-new 1 2)
<== f-6     (lista 4)
==> f-9     (lista)
==> f-10    (element 4)
==> Activation 0      R2: f-7,f-10
==> Activation 0      R2: f-5,f-10
<== f-5     (element 2)
<== Activation 0      R2: f-5,f-10
==> f-11    (lista-new 2 3)
<== f-7     (element 3)
==> f-12    (lista-new 3 4)
```

```
CLIPS> (facts)
f-0     (initial-fact)
f-8     (lista-new 1 2)
f-9     (lista)
f-10    (element 4)
f-11    (lista-new 2 3)
f-12    (lista-new 3 4)
For a total of 6 facts.
```

# Ejercicio (examen 2/11/2015, cuestión 5)

Dado la siguiente base de hechos:

(pila A B A A B B A pilaA pilaB)

que representa el estado inicial de un SBR, donde se tiene una pila inicial con bloques A y B y el objetivo es separar dichos bloques en dos pilas, una con bloques A y otra con bloques B.

Diseña la base de reglas.

```
(deffacts bh (pila A B A A B B A pilaA pilaB))
(defrule mover-a-pila-A
          (pila $?x A $?y pilaA   $?z) =>
  (assert (pila $?x   $?y pilaA A $?z)))
(defrule mover-a-pila-B
          (pila $?x B $?y pilaA $?z pilaB   $?zz) =>
  (assert (pila $?x   $?y pilaA $?z pilaB B $?zz)))
(defrule obj
          (pila pilaA $?z pilaB $?zz) =>
  (printout T "¡Solución encontrada!" crlf) (halt))
(set-strategy depth)
(watch facts)
(watch activations)
(reset)
(run)
(exit)
```

```
==> f-1     (pila A B A A B B A pilaA pilaB)
==> Activation 0      mover-a-pila-B: f-1
==> Activation 0      mover-a-pila-B: f-1
==> Activation 0      mover-a-pila-B: f-1
==> Activation 0      mover-a-pila-A: f-1
==> Activation 0      mover-a-pila-A: f-1
==> Activation 0      mover-a-pila-A: f-1
==> Activation 0      mover-a-pila-A: f-1
CLIPS> ==> f-2     (pila B A A B B A pilaA A pilaB)
==> Activation 0      mover-a-pila-B: f-2
==> Activation 0      mover-a-pila-B: f-2
==> Activation 0      mover-a-pila-B: f-2
==> Activation 0      mover-a-pila-A: f-2
==> Activation 0      mover-a-pila-A: f-2
==> Activation 0      mover-a-pila-A: f-2
==> f-3     (pila B A B B A pilaA A A pilaB)
==> Activation 0      mover-a-pila-B: f-3
==> Activation 0      mover-a-pila-B: f-3
==> Activation 0      mover-a-pila-B: f-3
==> Activation 0      mover-a-pila-A: f-3
==> Activation 0      mover-a-pila-A: f-3
==> f-4     (pila B B B A pilaA A A A pilaB)
==> Activation 0      mover-a-pila-B: f-4
==> Activation 0      mover-a-pila-B: f-4
==> Activation 0      mover-a-pila-B: f-4
==> Activation 0      mover-a-pila-A: f-4
==> f-5     (pila B B B pilaA A A A A pilaB)
==> Activation 0      mover-a-pila-B: f-5
==> Activation 0      mover-a-pila-B: f-5
==> Activation 0      mover-a-pila-B: f-5
==> f-6     (pila B B pilaA A A A A pilaB B)
==> Activation 0      mover-a-pila-B: f-6
==> Activation 0      mover-a-pila-B: f-6
==> f-7     (pila B pilaA A A A A pilaB B B)
==> Activation 0      mover-a-pila-B: f-7
==> f-8     (pila pilaA A A A A pilaB B B B)
==> Activation 0      obj: f-8
¡Solución encontrada!
```

# Ejercicio (examen 10/11/2014, cuestión 4)

Dado la siguiente base de hechos:

(numero 5)
(factorial 1)

que representa el estado inicial de un SBR para el cálculo del factorial de 5. Diseña la base de reglas.

```
(deffacts factorial (numero 5) (fact 1))
(defrule fact
  ?f1 <- (numero ?n1)
  ?f2 <- (fact ?n2)
  (test (> ?n1 1))
  =>
  (retract ?f1 ?f2)
  (assert (numero (- ?n1 1)))
  (assert (fact (* ?n2 ?n1 ))))
(set-strategy breadth)
(watch facts)
(watch activations)
(reset)
(run)
(exit)
```

```
CLIPS (V6.24 06/15/06)
CLIPS> (reset)
==> f-0     (initial-fact)
==> f-1     (numero 5)
==> f-2     (fact 1)
==> Activation 0       fact: f-1,f-2
CLIPS> (run)
<== f-1     (numero 5)
<== f-2     (fact 1)
==> f-3     (numero 4)
==> f-4     (fact 5)
==> Activation 0       fact: f-3,f-4
<== f-3     (numero 4)
<== f-4     (fact 5)
==> f-5     (numero 3)
==> f-6     (fact 20)
==> Activation 0       fact: f-5,f-6
<== f-5     (numero 3)
<== f-6     (fact 20)
==> f-7     (numero 2)
==> f-8     (fact 60)
==> Activation 0       fact: f-7,f-8
<== f-7     (numero 2)
<== f-8     (fact 60)
==> f-9     (numero 1)
==> f-10    (fact 120)
CLIPS> (exit)
```

# El problema del robot con un obstáculo

Se desea realizar un SBR que implemente el comportamiento de un robot simple que se mueve en una cuadrícula.

El robot solo se puede mover en 4 direcciones (arriba, abajo, dcha. e izq.), una casilla cada vez sin salirse de la cuadrícula ni moverse al obstáculo.

El tamaño de la cuadrícula se fija en los datos iniciales, así como la posición del obstáculo, de la meta y del robot inicialmente.

| | | | |
|---|---|---|---|
| 3 | | | Meta |
| 2 | | Obs. | |
| 1 | Robot | | |
| | 1 | 2 | 3 |

```
(deffacts bhrobot (grid 3 3) (obs 2 2) (meta 3 3) (robot 1 1))
(defrule derecha
  (robot ?x ?y)
  (obs ?ox ?oy)
  (grid ?gx ?gy)
  (test (< ?x ?gx))
  (test (not (and (eq (+ ?x 1) ?ox) (eq ?y ?oy))))
  =>
  (assert (robot (+ ?x 1) ?y)))
(defrule arriba
  (robot ?x ?y)
  (obs ?ox ?oy)
  (grid ?gx ?gy)
  (test (< ?y ?gy))
  (test (not (and (eq ?x ?ox) (eq (+ ?y 1) ?oy))))
  =>
  (assert (robot ?x (+ ?y 1))))
(defrule izquierda
  (robot ?x ?y)
  (obs ?ox ?oy)
  (test (> ?x 1))
  (test (not (and (eq (- ?x 1) ?ox) (eq ?y ?oy))))
  =>
  (assert (robot (- ?x 1) ?y)))
```

```
(defrule abajo
  (robot ?x ?y)
  (obs ?ox ?oy)
  (test (> ?y 1))
  (test (not (and (eq ?x ?ox) (eq (- ?y 1) ?oy))))
  =>
  (assert (robot ?x (- ?y 1)))
)

(defrule meta
  (declare (salience 10))
  (robot ?x ?y)
  (meta ?x ?y)
  =>
  (printout t "META ALCANZADA!" crlf)
  (halt)
)
;(set-strategy breadth)
;(watch facts)
;(watch activations)
;(reset)
;(run)
;(exit)
```

```
CLIPS> (reset)
==> f-0     (initial-fact)
==> f-1     (grid 3 3)
==> f-2     (obs 2 2)
==> f-3     (meta 3 3)
==> f-4     (robot 1 1)
==> Activation 0      arriba: f-4,f-2,f-1
==> Activation 0      derecha: f-4,f-2,f-1
CLIPS> (run)
==> f-5     (robot 2 1)
==> Activation 0      izquierda: f-5,f-2
==> Activation 0      derecha: f-5,f-2,f-1
==> f-6     (robot 3 1)
==> Activation 0      izquierda: f-6,f-2
==> Activation 0      arriba: f-6,f-2,f-1
==> f-7     (robot 3 2)
==> Activation 0      abajo: f-7,f-2
==> Activation 0      arriba: f-7,f-2,f-1
==> f-8     (robot 3 3)
==> Activation 10     meta: f-8,f-3
==> Activation 0      abajo: f-8,f-2
==> Activation 0      izquierda: f-8,f-2
META ALCANZADA!
```

# El problema de la agencia de transportes

Una agencia dispone de tres almacenes (A, B y C) con paquetes que debe transportar a otros almacenes con un camión que puede llevar un máximo de 10 paquetes.

Asume que en el estado inicial en el almacén A hay 4 paquetes para B y 3 para C, en el almacén B hay 7 paquetes para A y 3 para C, y en el almacén C hay 3 para A y 3 para B. Considera que el camión está en el almacén A y está vacío.

Diseña las siguientes reglas:

- ***Cargar*** paquetes siempre de diferente tipo de los ya cargados.
- ***Descargar*** paquetes.
- ***Mover*** camión entre almacenes.
- ***Parar*** cuando todos los paquetes han sido entregados.

```
(deffacts bhinicio (agencia almacen A destino B 4 destino C 3
  almacen B destino A 7 destino C 3
  almacen C destino A 3 destino B 3 camion A total 0))

(defrule cargar
  (agencia $?x almacen ?alm destino ?dest ?num $?y camion ?alm $?z
   total ?total)
  (test (not (member$ ?dest $?z)))
  (test (<= (+ ?total ?num) 10)) =>
  (assert (agencia $?x almacen ?alm $?y camion ?alm destino ?dest
   ?num $?z total (+ ?total ?num))))

(defrule descargar
  (agencia $?x camion ?loc $?y destino ?loc ?num $?z total ?tot) =>
  (assert (agencia $?x camion ?loc $?y $?z total (- ?tot ?num))))

(defrule mover
  (agencia $?x almacen ?alm $?y camion ?loc $?z) =>
  (assert (agencia $?x almacen ?alm $?y camion ?alm $?z)))

(defrule parar
  (agencia almacen A almacen B almacen C camion ? total 0) =>
  (printout t "PAQUETES ENTREGADOS!" crlf)
  (halt))
```

# El problema de las jarras de agua

Tenemos dos jarras de agua, X e Y, de 4 y 3 litros de capacidad.

No hay marcas de medida excepto la de capacidad máxima.

Inicialmente, ambas jarras están vacías.

El objetivo es tener 2 litros en X usando acciones del tipo:

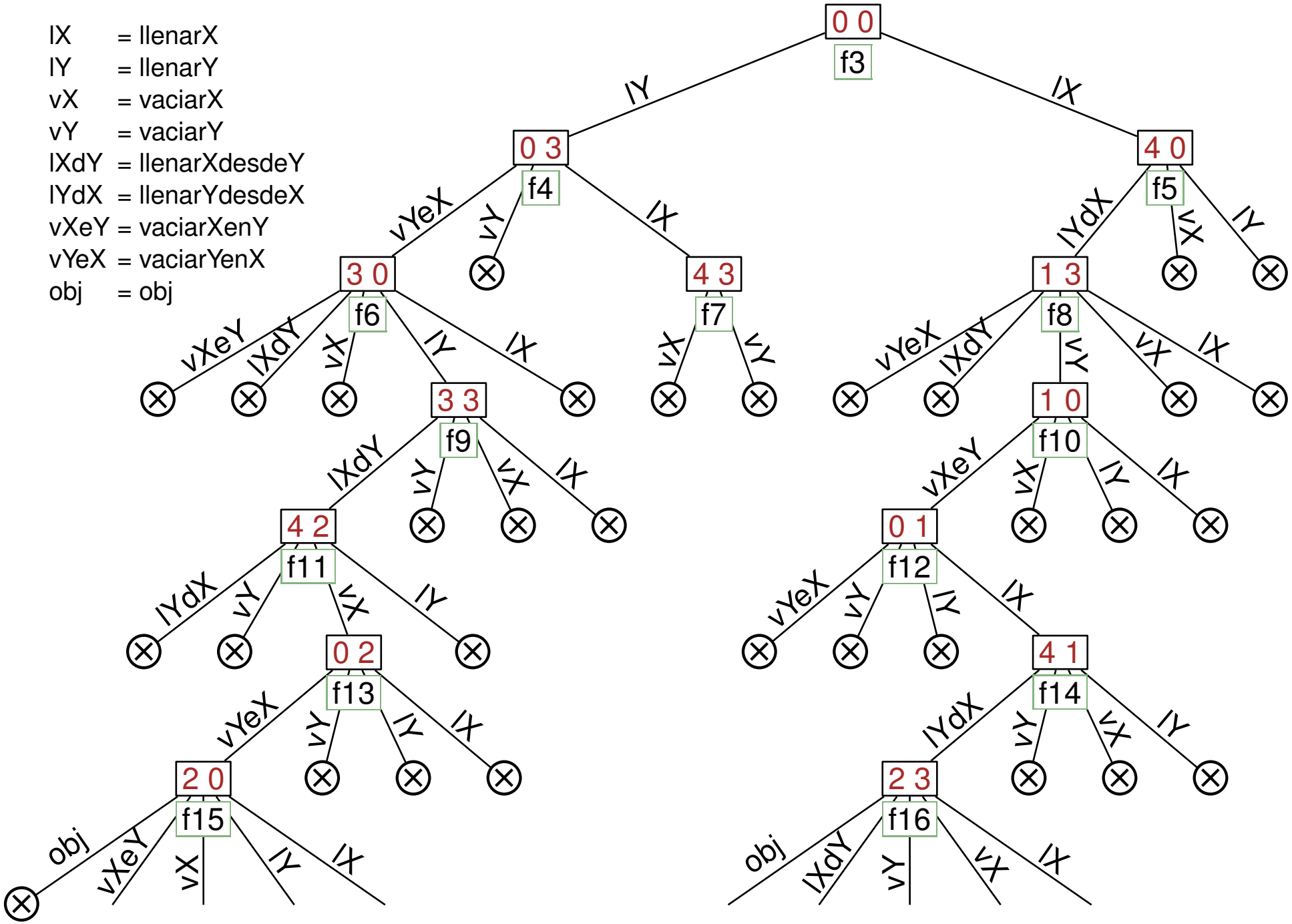
- Llenar X (Y).
- Llenar X desde Y (Y desde X).
- Vaciar X (Y).
- Vaciar X en Y (Y en X).

```
(deffacts bh (cap X 4) (cap Y 3) (agua X 0 Y 0))
(defrule llenarX (cap X ?capX) (agua X ?x Y ?y)
  (test (< ?x ?capX)) => (assert (agua X ?capX Y ?y)))
(defrule llenarY (cap Y ?capY) (agua X ?x Y ?y)
  (test (< ?y ?capY)) => (assert (agua X ?x Y ?capY)))
(defrule vaciarX (agua X ?x Y ?y)
  (test (> ?x 0)) => (assert (agua X 0 Y ?y)))
(defrule vaciarY (agua X ?x Y ?y)
  (test (> ?y 0)) => (assert (agua X ?x Y 0)))
(defrule llenarXdesdeY
  (cap X ?capX) (agua X ?x Y ?y)
  (test (> ?y 0)) (test (< ?x ?capX))
  (test (>= (+ ?x ?y) ?capX)) =>
  (assert (agua X ?capX Y (- ?y (- ?capX ?x)))))
(defrule llenarYdesdeX
  (cap Y ?capY) (agua X ?x Y ?y)
  (test (> ?x 0)) (test (< ?y ?capY))
  (test (>= (+ ?x ?y) ?capY)) =>
  (assert (agua X (- ?x (- ?capY ?y)) Y ?capY)))
(defrule vaciarXenY
  (cap Y ?capY) (agua X ?x Y ?y)
  (test (> ?x 0)) (test (<= (+ ?x ?y) ?capY)) =>
  (assert (agua X 0 Y (+ ?x ?y))))
(defrule vaciarYenX
  (cap X ?capX) (agua X ?x Y ?y)
  (test (> ?y 0)) (test (<= (+ ?x ?y) ?capX)) =>
  (assert (agua X (+ ?x ?y) Y 0)))
(defrule obj
  (agua X 2 Y ?) => (printout t "¡Solución encontrada!" crlf) (halt))
(set-strategy breadth)
(watch facts)
(watch activations)
(reset)
(run)
(exit)
```

```
CLIPS (V6.24 06/15/06)
CLIPS> ==> f-0     (initial-fact)
==> f-1     (cap X 4)
==> f-2     (cap Y 3)
==> f-3     (agua X 0 Y 0)
==> Activation 0      llenarY: f-2,f-3
==> Activation 0      llenarX: f-1,f-3
CLIPS> ==> f-4     (agua X 0 Y 3)
==> Activation 0      vaciarYenX: f-1,f-4
==> Activation 0      vaciarY: f-4
==> Activation 0      llenarX: f-1,f-4
==> f-5     (agua X 4 Y 0)
==> Activation 0      llenarYdesdeX: f-2,f-5
==> Activation 0      vaciarX: f-5
==> Activation 0      llenarY: f-2,f-5
==> f-6     (agua X 3 Y 0)
==> Activation 0      vaciarXenY: f-2,f-6
==> Activation 0      llenarYdesdeX: f-2,f-6
==> Activation 0      vaciarX: f-6
==> Activation 0      llenarY: f-2,f-6
==> Activation 0      llenarX: f-1,f-6
==> f-7     (agua X 4 Y 3)
==> Activation 0      vaciarY: f-7
==> Activation 0      vaciarX: f-7
==> f-8     (agua X 1 Y 3)
==> Activation 0      vaciarYenX: f-1,f-8
==> Activation 0      llenarXdesdeY: f-1,f-8
==> Activation 0      vaciarY: f-8
==> Activation 0      vaciarX: f-8
==> Activation 0      llenarX: f-1,f-8
==> f-9     (agua X 3 Y 3)
==> Activation 0      llenarXdesdeY: f-1,f-9
==> Activation 0      vaciarY: f-9
==> Activation 0      vaciarX: f-9
==> Activation 0      llenarX: f-1,f-9
==> f-10    (agua X 1 Y 0)
==> Activation 0      vaciarXenY: f-2,f-10
```

```
==> Activation 0       vaciarX: f-10
==> Activation 0       llenarY: f-2,f-10
==> Activation 0       llenarX: f-1,f-10
==> f-11    (agua X 4 Y 2)
==> Activation 0       llenarYdesdeX: f-2,f-11
==> Activation 0       vaciarY: f-11
==> Activation 0       vaciarX: f-11
==> Activation 0       llenarY: f-2,f-11
==> f-12    (agua X 0 Y 1)
==> Activation 0       vaciarYenX: f-1,f-12
==> Activation 0       vaciarY: f-12
==> Activation 0       llenarY: f-2,f-12
==> Activation 0       llenarX: f-1,f-12
==> f-13    (agua X 0 Y 2)
==> Activation 0       vaciarYenX: f-1,f-13
==> Activation 0       vaciarY: f-13
==> Activation 0       llenarY: f-2,f-13
==> Activation 0       llenarX: f-1,f-13
==> f-14    (agua X 4 Y 1)
==> Activation 0       llenarYdesdeX: f-2,f-14
==> Activation 0       vaciarY: f-14
==> Activation 0       vaciarX: f-14
==> Activation 0       llenarY: f-2,f-14
==> f-15    (agua X 2 Y 0)
==> Activation 0       obj: f-15
==> Activation 0       vaciarXenY: f-2,f-15
==> Activation 0       vaciarX: f-15
==> Activation 0       llenarY: f-2,f-15
==> Activation 0       llenarX: f-1,f-15
==> f-16    (agua X 2 Y 3)
==> Activation 0       obj: f-16
==> Activation 0       llenarXdesdeY: f-1,f-16
==> Activation 0       vaciarY: f-16
==> Activation 0       vaciarX: f-16
==> Activation 0       llenarX: f-1,f-16
¡Solución encontrada!
```

lX = llenarX
lY = llenarY
vX = vaciarX
vY = vaciarY
lXdY = llenarXdesdeY
lYdX = llenarYdesdeX
vXeY = vaciarXenY
vYeX = vaciarYenX
obj = obj



# Jarras de agua: traza simplificada