Dpto. Sistemas Informáticos y Computación
Escuela Técnica Superior de Ingeniería Informática
UNIVERSITAT POLITÈCNICA DE VALÈNCIA

SISTEMAS INTELIGENTES

PRÁCTICA 1

DISEÑO, IMPLEMENTACIÓN Y EVALUACIÓN DE UN SISTEMA BASADO EN REGLAS

El juego del 8-puzzle

Indice

1. Implementación del juego del 8-puzzle

En este apartado se presenta la implementación del juego del 8-puzzle como un SBR. Los elementos que intervienen en el problema son:

- **Base de Hechos inicial**: situación inicial de partida del puzzle
- **Estado Objetivo**: situación final del puzzle a la cual se quiere llegar
- **Reglas**: conjunto de posibles acciones a realizar (movimientos del cuadro vacío)
- **Estrategia de control**: búsqueda en anchura, búsqueda en profundidad y búsqueda heurística (se detallará en el siguiente apartado)

1.1 Patrones

Para la representación del puzzle se ha escogido un patrón ordenado donde representamos los valores de las 9 casillas ordenadas por filas. Un hecho asociado a este patrón representará un estado concreto del problema o configuración del 8-puzzle.

Por ejemplo, la siguiente configuración o estado del problema:

2	**8**	**3**
1	**6**	**4**
7		**5**

, se representará como:

```
(puzzle 2 8 3 1 6 4 7 0 5)
```

Los tres primeros valores corresponden a las casillas de la fila 1, los siguientes tres valores a los de la fila 2 y los últimos tres valores a los de la fila 3.

Se utilizará, adicionalmente, un conjunto de campos ó etiquetas adicionales:

- una etiqueta que indica el ***nivel de profundidad*** del hecho asociado. Esta etiqueta se empleará para dos objetivos distintos:
  - por una parte, para controlar que no se expanden nodos que superan el límite de profundidad prefijado por el usuario.
  - por otra parte, para devolver el límite de profundidad dónde la solución ha sido encontrada.

- una etiqueta para indicar cuál ha sido el ***último movimiento*** que ha originado tal estado. Este campo se empleará para evitar los ciclos directos en los movimientos del puzzle.

- una etiqueta para indicar el ***hecho sobre el que se aplicó el último movimiento*** para generar el estado actual. Este campo se empleará para recuperar el camino seguido para llegar hasta la solución.

Por tanto, un ejemplo de hecho inicial sería:

```
(puzzle 2 8 3 1 6 4 7 0 5 nivel 0 movimiento nulo hecho 0)
```

Y un ejemplo de hecho final sería:

```
(puzzle 1 2 3 8 0 4 7 6 5 nivel 5 movimiento derecha hecho <Fact-28>)
```

El campo **nivel** es un elemento constante y el elemento que aparece a continuación indica el nivel de dicho nodo en el árbol de búsqueda.

El campo **movimiento** es un elemento constante y el elemento que aparece a continuación guarda el movimiento que se ha realizado (izquierda, derecha, arriba, abajo) para llegar a dicho estado.

El campo **hecho** es un elemento constante y el elemento que aparece a continuación indica el índice del hecho sobre el que se aplicó el último movimiento para generar este estado.

Por tanto, la estructura de nuestro patrón sería:

(puzzle $x1^s$ $x2^s$ $x3^s$ $x4^s$ $x5^s$ $x6^s$ $x7^s$ $x8^s$ $x9^s$ nivel y^s movimiento z^s hecho w^s)

donde:

- **puzzle**, **nivel, movimiento** y **hecho** son cuatro campos constantes
- los campos que están etiquetados con un superíndice ‘s’ (single-valued) indican que es un campo mono-valuado; es decir, un elemento que debe ser necesariamente un único valor. Concretamente:
  - $xi^s \in [0\text{-}8] \; / \; \forall i,j \;\; xi \neq xj$
  - $y^s \in$ INTEGER
  - $z^s \in$ [arriba, abajo, derecha, izquierda]
  - $w^s \in$ fact-index

1.2 Reglas

Las reglas representan los movimientos que se pueden realizar en el puzzle. Como todo movimiento involucra una ficha del puzzle y la casilla en blanco, las reglas del problema se plantean como movimientos de la casilla en blanco. Por tanto, implementaremos los cuatro movimientos que se pueden realizar con la casilla que contiene el espacio en blanco, esto es, **derecha, izquierda, arriba** y **abajo**.

Desde este punto de vista, se podrá realizar un movimiento a la **derecha** siempre y cuando el espacio en blanco se encuentre en alguna de las posiciones indicadas por un punto negro en la siguiente figura:

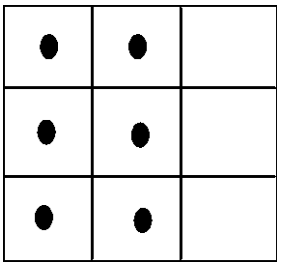

Fig. 1 Posiciones desde las cuales se puede realizar un movimiento a la derecha

Por tanto, tenemos que cada uno de los cuatro movimientos puede realizarse a partir de 6 posiciones distintas.

Las reglas implementan los cuatro posibles movimientos del espacio en blanco dentro del tablero. Cada una de las reglas instancia en su parte izquierda la configuración necesaria de un estado para poder mover el espacio en blanco en el sentido indicado por la regla. Por ejemplo, la regla `derecha` implementa cualquier posible movimiento del cuadro blanco hacia una posición que esté a su derecha. La regla `derecha` tiene el siguiente formato:

```
(defrule derecha
  ?f<-(puzzle $?x 0 ?y $?z nivel ?nivel movimiento ?mov hecho ?)
  (profundidad-maxima ?prof)
  (test (and (<> (length$ $?x) 2) (<> (length$ $?x) 5)))
  (test (neq ?mov izquierda))
  (test (< ?nivel ?prof))
=>
  (assert (puzzle $?x ?y 0 $?z nivel (+ ?nivel 1) movimiento derecha hecho ?f))
  (bind ?*nod-gen* (+ ?*nod-gen* 1)))
```

Fig. 2 Regla `derecha`

La parte izquierda de la regla se compone de dos patrones y tres tests:

1. el primer patrón instancia todos los hechos de la Base de Hechos correspondientes a las distintas configuraciones de los estados del puzzle, localizando la casilla en blanco en el elemento 0 y la casilla que está a su derecha en la variable `?y`.

2. el segundo patrón instancia un hecho estático que indica la profundidad máxima que se debe alcanzar en la búsqueda.

3. el primer test comprueba si el espacio en blanco en el patrón se encuentra en alguna de las 6 posiciones requeridas para mover el cuadro blanco a la derecha. Para ello se comprueba la longitud de la variable `$?x` que guarda todos los elementos a la izquierda de 0. Por ejemplo, si la casilla en blanco se encuentra en la posición [1,1] (casilla superior izquierda) entonces la longitud de la variable `$?x` será 0. En este caso, solo comprobamos que la longitud de la variable `$?x` no sea 2 ni 5, en cuyo caso la ficha en blanco estaría situada en las posiciones [1,3] y [2,3], respectivamente. No es necesario comprobar que la longitud de `$?x` sea

distinto de 8 (en cuyo caso la casilla blanca estaría en la posición [3,3]) porque esto no es posible a tenor de las variables del patrón.

4. el siguiente test comprueba que la configuración o estado que instancia el primer patrón no se haya generado por un movimiento a izquierdas (para evitar ciclos).

5. el último test se encarga de filtrar sólo aquellos hechos cuyo nivel de profundidad no excede el establecido por el usuario.

En la parte derecha de la regla se inserta el nuevo estado (hecho) que resulta como consecuencia de mover el espacio en blanco desde la posición en la que éste se encuentra hacia la derecha. Por tanto, el nuevo hecho intercambia las posiciones de la casilla en blanco (0) con la casilla que está a su derecha `(?y)`.

Para las reglas `arriba` y `abajo` hay que tener en cuenta que se tratan de movimientos verticales en una estructura lineal por lo que hay que localizar la ficha con la que se intercambia la casilla en blanco.

El código de la regla abajo sería:

```
(defrule abajo
  ?f<-(puzzle $?x 0 ?a ?b ?c $?z nivel ?nivel movimiento ?mov hecho ?)
 (profundidad-maxima ?prof)
 (test (neq ?mov arriba))
 (test (< ?nivel ?prof))
=>
 (assert (puzzle $?x ?c ?a ?b 0 $?z nivel (+ ?nivel 1) movimiento abajo
hecho ?f))
 (bind ?*nod-gen* (+ ?*nod-gen* 1)))
```

Fig. 3 Regla `abajo`

La casilla blanca se intercambia siempre con la ficha que ocupa la posición de la variable `?c`. Se puede observar que al poner tres variables mono-valuadas (`?a ?b ?c)` después del 0, estamos exigiendo que haya necesariamente tres elementos, lo cual evitaría los movimientos hacia abajo que no son posibles (cuando el espacio blanco está en la última fila).

1.3 Datos iniciales, objetivo y solución

El estado **objetivo** con el que se trabajará en el problema del puzzle es el siguiente:

1	**2**	**3**
8		**4**
7	**6**	**5**

El estado objetivo se representa directamente la regla `objetivo` que se muestra en la figura 4.

```
(defrule objetivo
     (declare (salience 100))
   ?f <-(puzzle 1 2 3 8 0 4 7 6 5 nivel ?n movimiento ?mov hecho ?)
  =>
   (printout t "SOLUCION ENCONTRADA EN EL NIVEL " ?n crlf)
   (printout t "NUM DE NODOS EXPANDIDOS O REGLAS DISPARADAS " ?*nod-gen* crlf)
   (printout t "HECHO OBJETIVO " ?f crlf)
   (halt))
```

Fig. 4 Regla `objetivo`

Esta regla tiene una prioridad máxima (`salience 100`) dado que es la regla que detecta que se ha llegado al estado objetivo. En el momento que se produzca un hecho que unifique con la configuración objetivo, la regla objetivo se disparará automáticamente concluyendo así el proceso inferencial. El comando (`halt`) detiene el Motor de Inferencia.

También se define una regla con una prioridad inferior al resto de reglas para detectar el caso en el que no se encuentra solución al problema.

```
(defrule no_solucion
    (declare (salience -99))
    (puzzle $? nivel ?n $?)
=>
    (printout t "SOLUCION NO ENCONTRADA" crlf)
    (printout t "NODOS GENERADOS: " ?*nod-gen* crlf)
    (halt))
```

Fig. 5 Regla `no_solucion`

Adicionalmente, tenemos una función `inicio` encargada de preguntar por la profundidad máxima con la que se desea trabajar y el tipo de estrategia a emplear (de momento, anchura y profundidad). Dependiendo de la respuesta, se activa la estrategia de la agenda correspondiente, ya que la estrategia de control (anchura/profundidad) la implementa la propia agenda de CLIPS. Además, se inserta en la base de hechos el estado inicial y un hecho estático que indica el máximo nivel de profundidad a alcanzar en la búsqueda.

```
(deffunction inicio ()
 (reset)
 (printout t "Profundidad Maxima:= " )
 (bind ?prof (read))
 (printout t "Tipo de Búsqueda " crlf "1.- Anchura" crlf
             "2.- Profundidad" crlf )
 (bind ?a (read))
 (if (= ?a 1)
       then (set-strategy breadth)
       else (set-strategy depth)))
  (assert (puzzle 2 8 3 1 6 4 7 0 5 nivel 0 movimiento nulo hecho 0))
  (assert (profundidad-maxima ?prof))
)
```

Fig. 6 Función `inicio`

Por último, se dispone de la función `camino`, que muestra por pantalla la secuencia de movimientos aplicados (y sobre qué hecho) para alcanzar la solución. Se invoca de la siguiente forma: `(camino <fact-index>)`.

Por ejemplo, si tras la ejecución de una configuración del puzzle, obtenemos una salida por pantalla del siguiente tipo (ver regla objetivo en figura 4):

```
SOLUCION ENCONTRADA EN EL NIVEL 5
NUMERO DE NODOS EXPANDIDOS O REGLAS DISPARADAS 36
HECHO OBJETIVO <Fact-39>
```

, para recuperar la solución de 5 pasos de la configuración probada, debemos simplemente ejecutar `(camino 39)`.

1.4 Configuraciones resolubles

En el problema del puzzle, puede darse el caso no exista solución para una configuración determinada. Por ejemplo, dado el estado inicial:

2	**1**	**3**
8		**4**
7	**6**	**5**

no existe combinación de movimientos derecha-izquierda-arriba-abajo que consiga llegar al estado final descrito en la sección anterior. En este caso, se dice que la configuración es irresoluble.

Con el objeto de comprobar si una configuración particular es resoluble o no (es decir, si es posible o no encontrar un camino que conduzca al estado objetivo), se proporciona la función `comprobar_conf` en el fichero `auxiliar.clp`. Esta función recibe como argumento una lista que representa la configuración del puzzle que se desea comprobar y devuelve `TRUE` si la configuración es resoluble y `FALSE` en caso contrario.

Por ejemplo:

```
CLIPS> (comprobar_conf (create$ 2 8 3 1 6 4 7 0 5))
TRUE

CLIPS> (comprobar_conf (create$ 2 1 3 8 0 4 7 6 5))
FALSE
```

El fichero `auxiliar.clp` debe cargarse en CLIPS siempre que se quiera realizar la comprobación de una determinada configuración del puzzle.

2. Estrategias de control *anchura* y *profundidad*

A través de las estrategias de control anchura y profundidad, implementamos una búsqueda en anchura/profundidad en árbol. En este punto, conviene mencionar un aspecto sobre el control de nodos repetidos en CLIPS:

1. En términos generales, CLIPS realiza un control automático de nodos repetidos porque, por defecto, no permite duplicidad de hechos. Esto significa que si se intenta insertar un nodo idéntico a otro que ya existe en la BH, CLIPS no lo insertará.
2. Sin embargo, en nuestro modelo utilizamos un campo **nivel** que es la profundidad de un nodo. Esto implica que un nodo de nivel 3 que represente el mismo estado que un nodo de nivel 5 no estarán representados por los mismos hechos y, por tanto, CLIPS insertará ambos nodos en la BH. Además, se incluye el campo **hecho** (para la recuperación del camino), lo que supone un factor totalmente distintivo entre dos configuraciones idénticas del puzzle.
3. En términos prácticos, por lo expuesto en 2, no existe control de nodos repetidos en CLIPS, excepto el control de ciclos directos que se realiza a través del campo **movimiento**.
4. No obstante, si no se mantuviera el **nivel** ni el **movimiento** del nodo ni el **hecho padre**, entonces CLIPS realizaría un control automático de los nodos repetidos. Pero, en dicho caso, no podríamos conocer el nivel de profundidad de la solución ni recuperar el camino para alcanzar la solución.

A continuación se muestra un par de iteraciones del problema del puzzle con el funcionamiento de la agenda en Anchura y su equivalencia en un proceso de búsqueda en árbol.

Inicialmente, la situación es la que se muestra en la siguiente pantalla, que se corresponde con el estado del proceso de búsqueda de la figura siguiente.

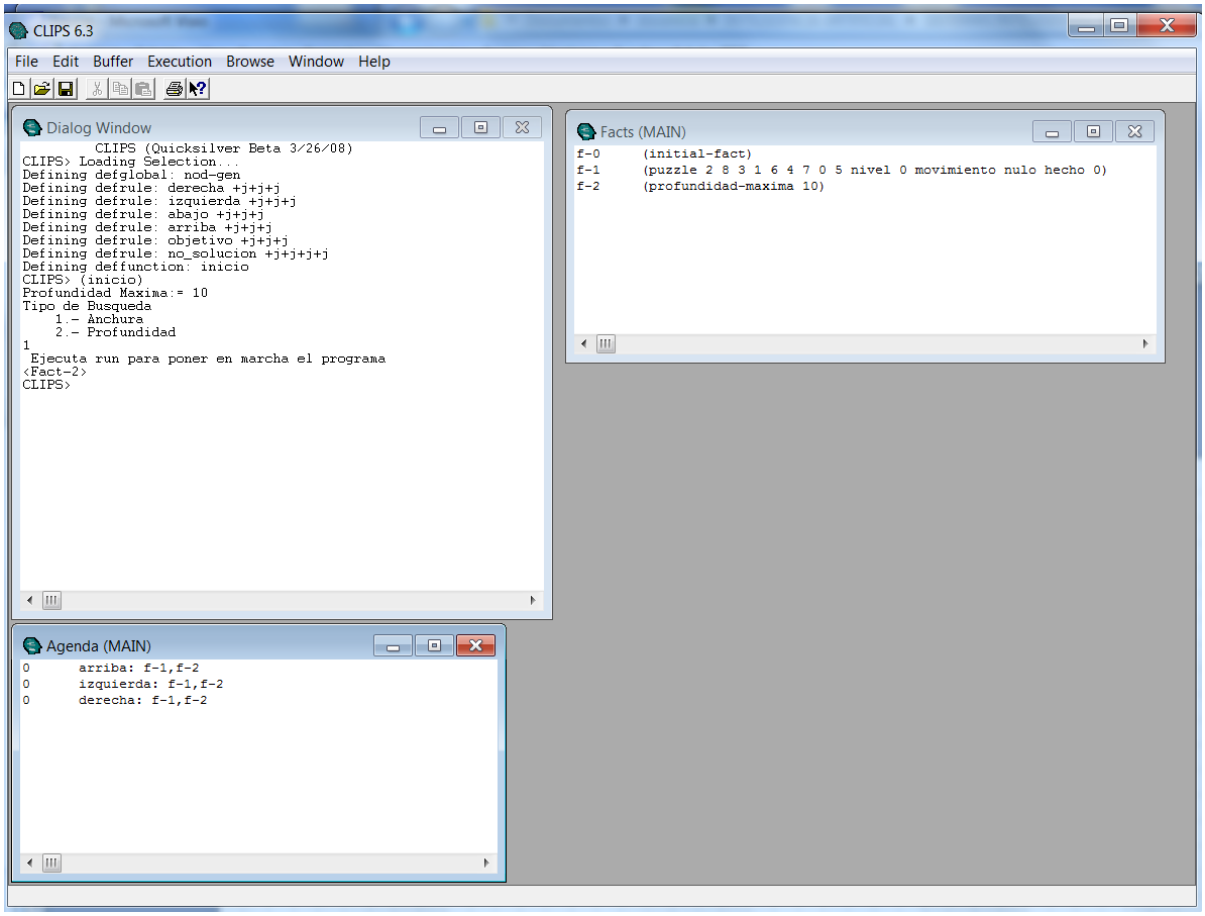

Fig. 7 Situación inicial de la Agenda y Base de Hechos

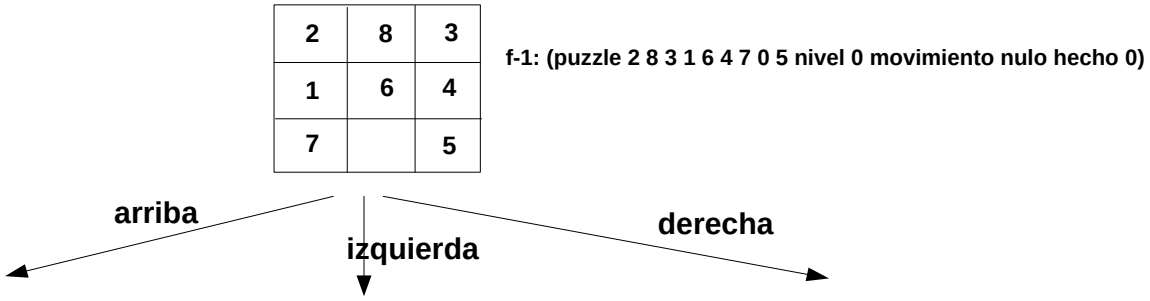

Fig. 8 Estado del proceso de búsqueda correspondiente a la Fig. 7

Las tres activaciones de la Agenda representan los tres posibles movimientos que se pueden aplicar en la configuración inicial. A diferencia de un proceso de búsqueda clásico, los nodos sucesores aún no se han generado (solo está el hecho `f-1` en la Base de Hechos).

Si ejecutamos un paso del proceso inferencial, obtenemos:

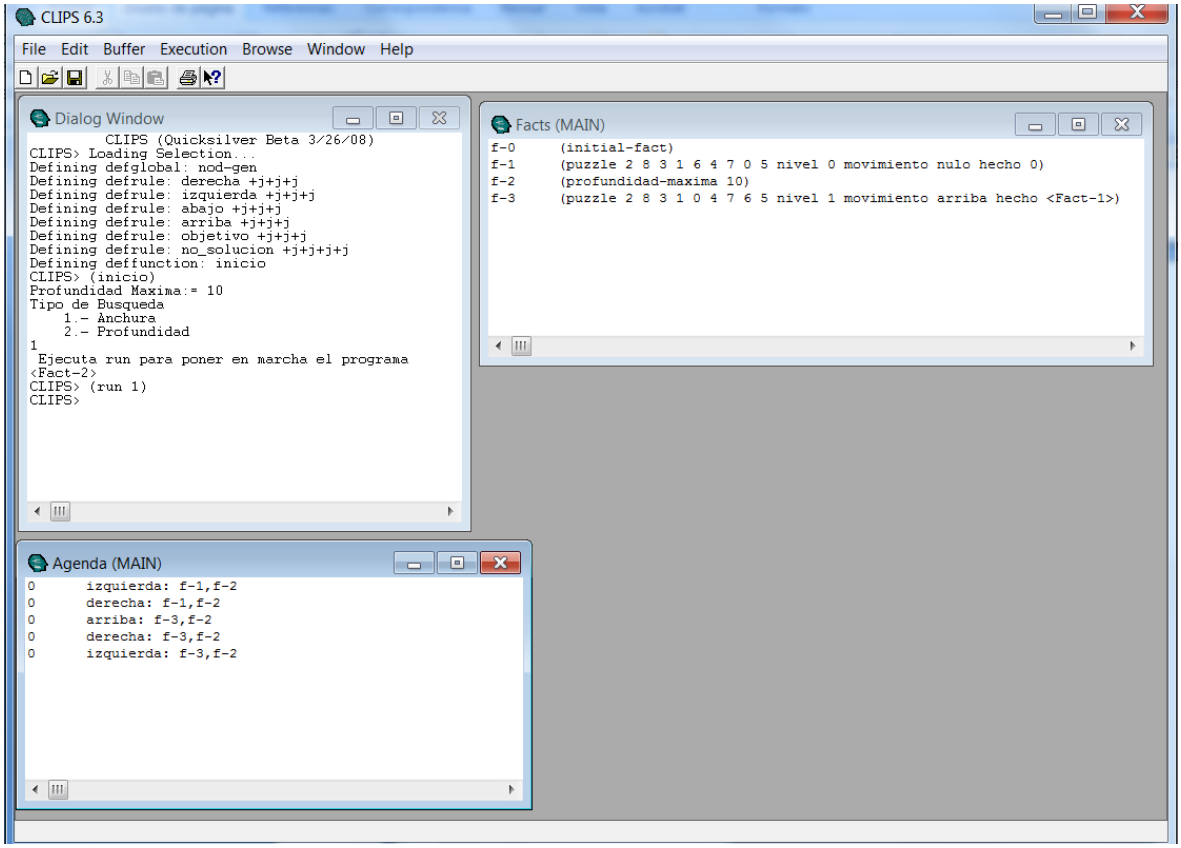

Fig. 9 Situación de la Agenda y BH después de un ciclo inferencial

, que se corresponde con:

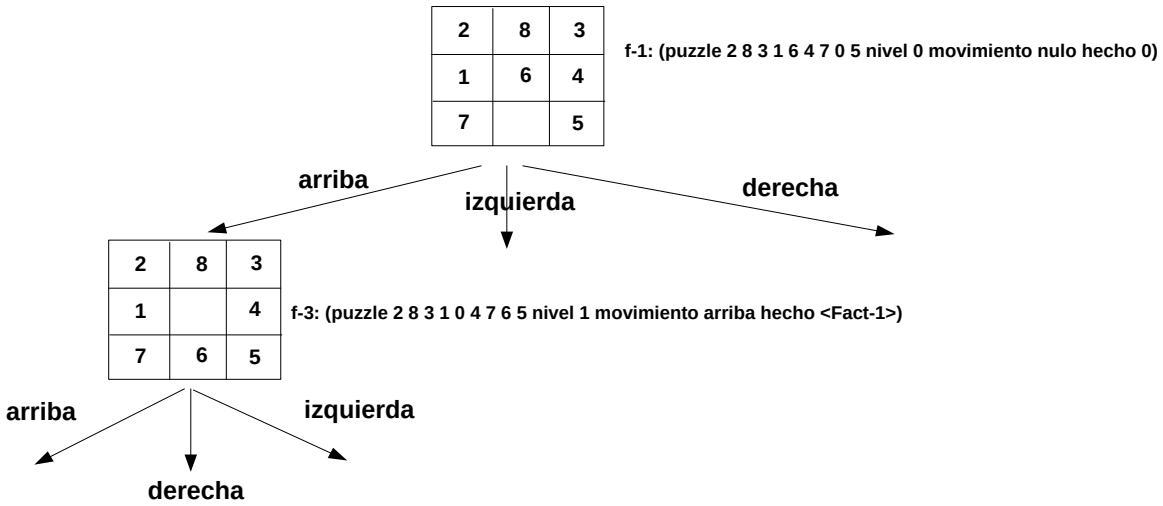

Fig. 10 Estado del proceso de búsqueda correspondiente a la Fig. 9

A continuación se dispararía la regla `izquierda` del nodo raíz, y luego la regla `derecha`, y luego la regla `arriba` del nodo representado con el hecho `f-3`, y así sucesivamente, implementando un proceso de búsqueda en anchura. Cuando finaliza el proceso de búsqueda, obtendríamos una situación como la que se muestra en la figura 11.

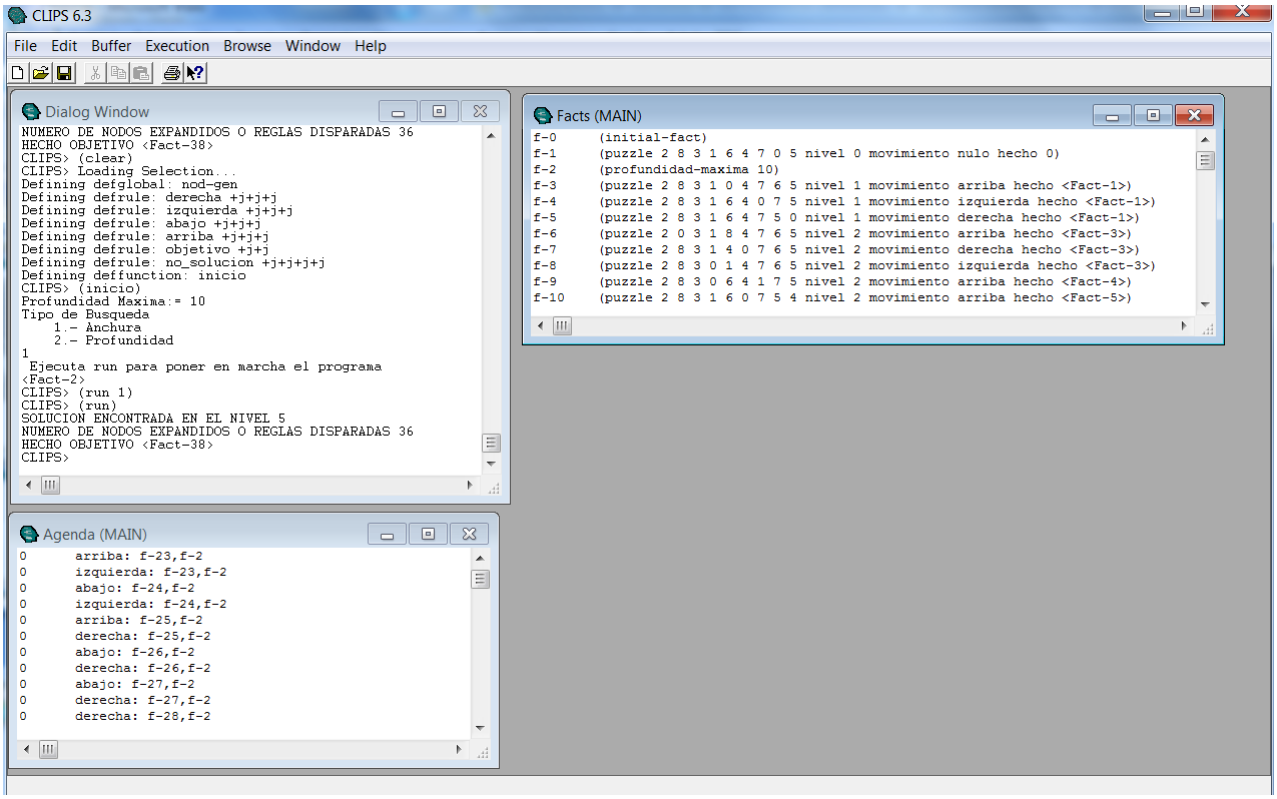

Fig. 11 Situación de la Agenda y BH al finalizar la ejecución

En este caso, si utilizamos la función `(camino 38)`, donde 38 es el número del hecho objetivo, obtendríamos esta salida:

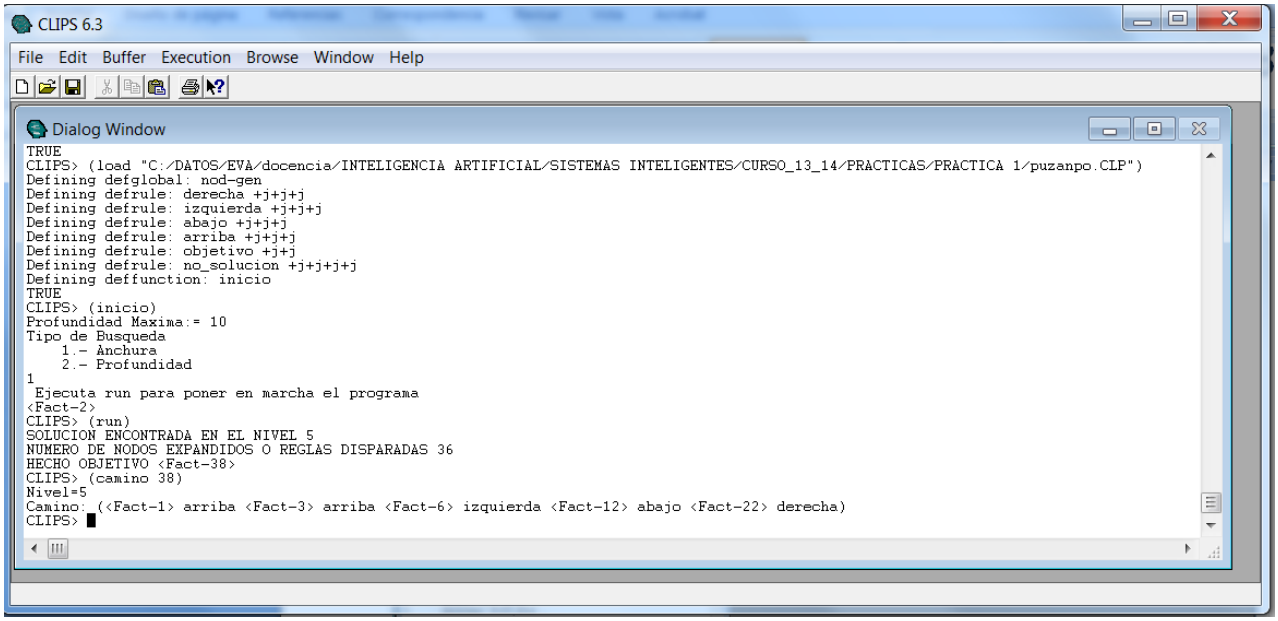

Fig. 12 Obtención del camino solución para el puzzle de la figura 8

3. Ejecución del sistema basado en reglas

Se proporciona un conjunto de ficheros:

- `puzanpo.clp`: corresponden a diferentes implementaciones del sistema de producción del puzzle: anchura, profundidad y con la heurística descolocadas, respectivamente.

- `auxiliar.clp`: contiene la función `comprobar_conf` para comprobar si la configuración que se pasa como parámetro es resoluble.

Para realizar una ejecución, se cargará el fichero correspondiente al sistema basado en reglas (`puzanpo.clp`), modificando la configuración inicial del problema en la función `inicio`. Previamente, se puede comprobar mediante la función del fichero `auxiliar.clp` si la configuración del puzzle a probar tiene solución.

4. Pruebas de evaluación

Para la ejecución de una configuración particular del puzzle, modificaremos el estado inicial en la función inicio del SBR correspondiente. Por ejemplo:

8	**1**	**3**
7	**2**	**5**
4		**6**

La situación inicial de este puzzle se corresponde con el siguiente hecho:

```
(puzzle 8 1 3 7 2 5 4 0 6 nivel 0 movimiento nulo hecho 0))
```

Los resultados para diferentes ejecuciones, tanto en anchura, profundidad como con búsqueda heurística y con diferentes niveles de profundidad, son los siguientes:

Estrategia	*Nivel de profundidad máximo*	*Nivel de la solución*	*Número de nodos expandidos*
Anchura	10	9	442
Anchura	20	9	442
Profundidad	10	9	479
Profundidad	20	15	68318