Práctica 6: Representación de conocimiento en Prolog mediante reglas
Inteligencia Artificial

Autores (Grupo 5):
José Javier Cortés Tejada
Pedro David González Vázquez

1. Parte 1: sistema de reglas para parentescos

Los predicados correspondientes a la madre, el hijo y la hija han sido contruidos en base a los hechos proporcionados en la práctica, mientras que el resto de predicados han sido construidos en base a los ya mencionados, sin usar ningún hecho.

2. Parte 2: puzzle blancas y negras

2.1. Representación del problema

De cara al desarrollo de la segunda parte de la práctica hemos optado por definir el problema en Prolog usando listas, de manera que los estados quedan representados por dos listas donde una de ellas contiene las piezas que están a la izquierda de la posición vacía, mientras que la otra contiene las piezas que están a la derecha de la posición vacia, obteniendo una representación como la siguiente:

$$\textbf{state}([' N',' N',' N'],[' B',' B',' B'])$$

2.2. Operaciones

En cuanto a los operandos disponibles, estos siguen una configuración como la siguiente:

move(state(L, R), state(FL, FR), 'xxx')

donde **state(L, R)** es el estado inicial sobre el que se ejecuta el movimiento, **state(FL, FR)** es el estado resultado de aplicar el movimiento y 'xxx' es el tipo de movimiento que se realiza sobre el tablero. Este puede ser:

- **iii**: una ficha salta otras dos hacia la derecha.
- **ii**: una ficha salta otra ficha hacia la derecha.
- **i**: una ficha se mueve a un hueco adyacente hacia la derecha.
- **ddd**: una ficha salta otras dos hacia la izquierda.
- **dd**: una ficha salta otra ficha hacia la izquierda.
- **d**: una ficha se mueve a un hueco adyacente hacia la izquierda.

En cuanto a la codificación de los operadores, cabe destacar que solamente en los que desplazan fichas hacia la derecha, como en el caso siguiente:

```
move(state(L, R), state(FL, FR), 'iii'):-
 reverse(L, [A,B,C|S]),
 append([B, A, C], R, FR),
 reverse(S, TEMP),
 append(TEMP,[] ,FL).
```

se ha usado **reverse** para acceder al elemento de las listas que queremos permutar, mientras que en aquellas donde movemos fichas hacia la izquierda simplemente usamos **append** sobre las listas del estado resultado para devolver las fichas en el orden que nos interesa, pues en función de la longitud del desplazamiento hemos definido la segunda lista del estado de partida como una lista con 1, 2 o 3 elementos en la cabeza, en función de nuestras necesidades:

```
move(state(L, [A, B, C| S]), state(FL, FR), 'ddd'):-
 append(L, [C, A, B], FL),
 append(S, [] ,FR).
```

A la hora de concatenar los elementos a su lista objetivo, estos han de intercambiar su cabeza y su último elemento de la cola, para así realizar el swap.

También remarcar que la no existencia de precondición al uso, eso es, una sentencia que compruebe por si misma la validez del movimiento es intencionada, pues las cadenas no validas ya serán descartadas al encajar el patrón, pues obligamos a tener a dichas listas a tener al menos un número suficiente de elementos para que el movimiento sea valido

3. Diferencias entre Jess y Prolog

Ambos lenguajes entra dentro del grupo de lenguajes declarativos, aunque podemos destacar alguna que otra diferencia bastante interesante:

- Prolog se centra mucho más en dar una solución a un problema, mientras que Jess está más enfocado a dar respuesta a varias entradas.
- Los resultados que obtenemos en Jess son almacendos en la Memoria de Trabajo (MT), mientras que en Prolog, si queremos saber de nuevo la solución a un problema debemos calcularlo de nuevo.
- En Jess se emplea el encadenamiento hacia delante, donde aplicamos sobre los hechos una serie de reglas para inferir nuevo conocimiento mientras que en Prolog tenemos encadenamiento hacia atrás, donde partimos de una hipótesis y tratamos de demostrarla usando la información que poseemos.

4. Algoritmo aplicado en resolución

Prolog emplea un algoritmo de búsqueda en profundidad, de manera que la demostración de las hipótesis iniciales se basa en partir de ésta e ir aplicando predicados que encajen de tal forma que por cada coincidencia se abre una rama nueva en el árbol de exploración, la cual se recorre completamente para ver si se tiene éxito o no.