Problema de las Jarras de Agua en CLIPS

Pablo Rodríguez Guillén (Representante): 32733455Q
José Márquez Doblas: 46074763J

10 de mayo de 2019

Índice

1. Enunciado del Problema

El objetivo de este trabajo es la implementación de las técnicas de búsqueda primero en profundidad y primero en anchura para solucionar un problema típico de Inteligencia Artificial. El problema de las jarras de agua. El planteamiento del problema es el siguiente:

Disponemos de dos jarras de agua, una de 4 litros de capacidad y otra de 3 litros de capacidad. Inicialmente están ambas vacías. El estado objetivo es que la jarra de 4 litros de capacidad contenga dos litros de agua, independientemente el contenido de la otra, sabiendo que en ninguna de las jarras hay una señal de volumen distinta de su capacidad. Para conseguir este objetivo, podemos realizar las siguientes acciones:

1. Llenar la jarra de 4 litros completamente (para ello, la jarra de 4 litros no debe estar completamente llena).
2. Llenar la jarra de 3 litros completamente (para ello, la jarra de 3 litros no debe estar completamente llena).
3. Vaciar la jarra de 4 litros (para ello, la jarra debe contener algo de liquido).
4. Vaciar la jarra de 3 litros (para ello, la jarra debe contener algo de liquido).
5. Verter el contenido de la jarra de 4 litros en la jarra de 3 litros (para ello, la jarra de 4 litros debe contener algo de liquido y la de 3 litros no estar completamente llena).
6. Verter el contenido de la jarra de 3 litros en la jarra de 4 litros (para ello, la jarra de 3 litros debe contener algo de liquido y la de 4 litros no estar completamente llena).

Identificar y representar los hechos necesarios para resolver el problema en clips mediante las dos técnicas de búsqueda comentadas anteriormente. Ha de controlarse también que no haya nodos repetidos.

2. Estrategia empleada

2.1. Representación de la información

Para representar la información del problema hemos utilizado hechos no ordenados, utilizando una plantilla *node* a través del constructor *deffacts*. La plantilla cuenta con tres atributos:

- ***multisolt jars***: durante todo el programa contendrá un hecho ordenado con los siguientes 4 sibolos: *j3 x j4 y* siendo x e y los valores que representan el contenido de la jarra de cuatro litros y de tres litros. Estos valores van a oscilar entre cero y cuatro, dado que la lógica implementada en las reglas no lo permite de otro modo.
- ***slot father***: contiene un hecho del tipo *FACT-ADRESS SYMBOL*, es decir, la dirección de memoria de otro hecho. En este caso, será un hecho ordenado *node* que apuntará al nodo padre del nodo en cuestión. Hay un símbolo permitido *none*, para poder representar a la raíz del árbol del problema, que no tiene nodo padre.
- ***slot level***: contiene un número entero que representa el nivel del profundidad del nodo en el árbol. La profundidad de un nodo es la profunidad del padre más uno. La profundidad del nodo raíz es 0.

2.2. Análisis de diseño e implementación

En los siguientes apartados se detalla la estrategia concreta seguida para los métodos de búsqueda primero en anchura y primero en profundidad. A un nivel más general hemos implementado reglas que se podrían dividir en dos grupos.

- Reglas de las operaciones permitidas por el enunciado para generar nuevos estados:
  - **Llenado de jarras:** Se permite siempre que la jarra a llenar no esté llena de antemano.
  - **Vaciado de jarras:** Se permite siempre que la jarra a vaciar no esté vacía de antemano.
  - **Vertido de una jarra en otra:** Existen dos versiones de la regla para cada jarra. Esto es porque el comportamiento de la regla varía si al verter una jarra en otra se derrama el contenido, lo cual no se permite en el problema. Para solucionar esto planteamos dos soluciones, utilizar funciones procedurales de tipo *if* o implementar dos reglas. Optamos por la segunda opción ya que se nos recomendaba usar el menor número de funciones procedurales posibles.
- Reglas para controlar el flujo del programa:
  - ***initial*:** Se ejecuta siempre al principio, ya que el único hecho en la base de hechos al inicio del programa es el *initial-fact*. Afirma el hecho correspondiente al nodo raíz del árbol del problema.
  - ***removeTwins*:** Por cada nodo, se generan todos los hijos posibles. Esto causa que estados que ya hayan aparecido anteriormente durante la ejecución del programa se vuelvan a crear. Para evitar esto, la regla *removeTwins* tiene una prioridad muy alta y sus antecedentes se comprueban cada vez que se genera un nuevo nodo. Si el nuevo nodo es uno repetido, se activa la regla y se elimina el nodo.

- ***finalResult*:** Al igual que *removeTwins* tiene una prioridad muy alta, sus antecedentes comprueban si el nodo en cuestión tiene la jarra de cuatro litros con dos litros. Si se activa, el programa muestra el contenido de la base de hechos con el comando *(facts)* y finaliza.

2.2.1. Búsqueda primero en anchura

Para que el programa haga una búsqueda primero en anchura, hay que asegurar que no se incremente el nivel con el que se generan los nuevos nodos, hasta que no se hayan generado todos los nodos de un mismo nivel.

Para controlar esto, hemos definido un hecho *globalLevel n* donde *n* es el nivel de los padres que están generando nuevos nodos en un momento determinado de la ejecución de nuestro programa. Estos nuevos nodos, como es lógico, incrementan su nivel en una unidad. Para asegurarnos de que esto es así, todas las reglas que generan nuevos nodos del archivo *JarrasAnchura.clp* tienen un antecedente que comprueba que el nivel del nodo que activa la función es igual al que alberga *globalLevel*.

Cuando todos los nodos de un mismo nivel han generado todos sus hijos posibles (recordamos que cada vez que se genera un nuevo nodo se pueden activar las reglas de *removeTwins* y *finalResult*) activando las reglas que generan nuevos nodos, se activa una función de menor prioridad que solo hemos implementado en la búsqueda primero en anchura que simplemente incrementa *globalLevel* en una unidad, lo que permite que los estados de nivel mayor, puedan activar reglas.

2.2.2. Búsqueda primero en profundidad