TP6-CLIPS

LENGUAJES DE REGLAS Y PROPAGACIÓN DE RESTRICCIONES

Asignatura: Inteligencia Artificial
Fecha: 22/11/2020
Autor: Diego Marco Beisty, 755232

Primera tarea

Problema 1 CLIPS (fichero fichas.clp)

Se ha resuelto el problema de las fichas utilizando el lenguaje de reglas CLIPS.
La estrategia que se ha utilizado es la búsqueda A* utilizando la heurística de fichas desordenadas.
La solución consta de 4 módulos:
El módulo “main” donde se ha definido la plantilla nodo, los estados inicial y final, y la función heurística de fichas descolocadas.
El módulo “restricciones” declara una regla que permite eliminar aquellos nodos generados que ya se habían visitado anteriormente.
El módulo operadores define dos reglas. La regla “moverDerecha” se sensibiliza para tres casos posibles en los que una ficha se mueve hacia el hueco saltando a la derecha sobre cero fichas, sobre una ficha o sobre dos fichas.
La regla “moverIzquierda” es equivalente a “moverDerecha” cuando una ficha salta hacia el hueco moviéndose a la izquierda, sobre cero, una o dos fichas.
El módulo solución declara una regla que se sensibiliza con el nodo objetivo y otra regla para mostrar el número de pasos y la traza del camino solución.

Cabe destacar que para controlar qué reglas se pueden sensibilizar en cada momento se han apilado las agendas de los módulos correspondientes. Concretamente tras detectar en el módulo main, el mejor nodo encontrado con la estrategia A* se cierra. Posteriormente se apila la agenda del módulo “operadores” para generar sus nodos abiertos sucesores. Cuando ya no se sensibiliza ninguna regla del módulo operadores, se desapila su agenda y se vuelven a sensibilizar las reglas del módulo main.

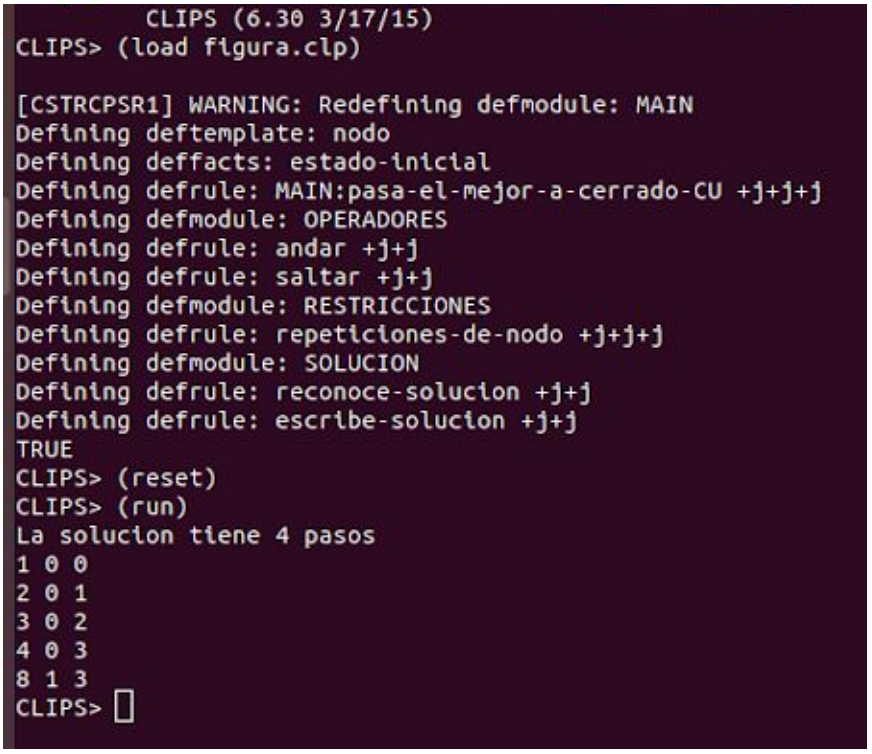
Tras ejecutar varias veces el programa, se observa que el estado objetivo se encuentra siempre a distancia 10
A continuación se muestra la traza del camino solución:

```
         CLIPS (6.30 3/17/15)
CLIPS> (load fichas.clp)

[CSTRCPSR1] WARNING: Redefining defmodule: MAIN
Defining deftemplate: nodo
Defining defglobal: estado-inicial
Defining defglobal: estado-final
Defining deffunction: heuristica
Defining defrule: inicial +j+j
Defining defrule: MAIN:pasa-el-mejor-a-cerrado-A* +j+j+j
Defining defmodule: RESTRICCIONES
Defining defrule: repeticiones-de-nodo +j+j+j
Defining defmodule: OPERADORES
Defining defrule: moverDerecha +j+j
Defining defrule: moverIzquierda +j+j
Defining defmodule: SOLUCION
Defining defrule: reconoce-solucion +j+j
Defining defrule: escribe-solucion +j+j
TRUE
CLIPS> (run)
La solucion tiene 10 pasos
B B B H V V V
B H B B V V V
B V B B H V V
B V H B B V V
B V V B B H V
B V V B B V H
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 H B B B
CLIPS>
```

Problema 3 CLIPS (fichero figura.clp)

Se ha resuelto el problema de la figura mediante el lenguaje de reglas CLIPS.
Se ha seguido el mismo procedimiento que en el problema 1 adaptando la representación del estado, las operaciones posibles en cada estado y la detección del nodo objetivo.clips.

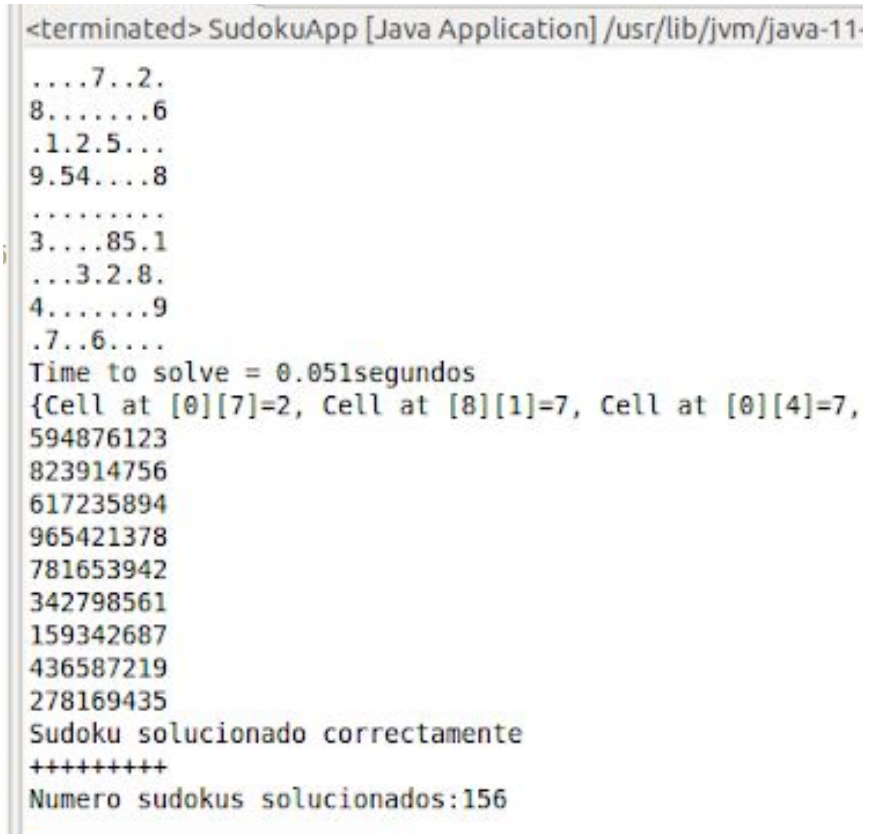


Segunda tarea

Se ha implementado el programa “SudokuApp” que resuelve sudokus mediante propagación de restricciones.
Otras clase implementada es “SudokuVariable” que representa una celda de sudoku con sus coordenadas x e y y su valor. También “SudokuConstraint” que añade la condición de que dos variables no puedan ser iguales. para que posteriormente se puedan añadir las restricciones de que no pueden existir variables con un mismo valor tanto en una fila, columna y cuadrado.

Después de estudiar la clase “MapColoringApp” se ha decidido utilizar la estrategia “ImprovedBacktrackingStrategy” puesto que es la que mejor resultado obtiene en tiempo.

A continuación se muestra el último sudoku resuelto de los 156 (ficheros easy50.txt, hardest.txt y top95.txt):



Tercera tarea

Se ha implementado el programa “NQueensMinConflictApp” que resuelve el problema de las 8 reinas mediante propagación de restricciones y búsqueda local.
Se han adaptado los ficheros de la segunda tarea para implementar las clases “NQueensConstraint”, “NQueensVariable” y “NQueensProblem”.
EN “NQueensVariable” se representa una reina localizada en una columna “y” del tablero. Inicialmente tiene asignada la fila cero.
El dominio de la variable es el rango [0,7] que representa el número de fila en el que puede estar en el tablero.
En “NQueensConstraint” se indica que dos variables no pueden pertenecer ni a la misma fila, ni a la misma columna, ni a la misma diagonal.

El algoritmo MQueensMInConflict resuelve el problema CSP mediante búsqueda local. Para seleccionar el siguiente estado tiene en cuenta que el número de restricciones que no se cumplen sea el menor. De esta forma el estado objetivo será aquel que cumpla todas las restricciones.
Se ha observado que esta estrategia no siempre encuentra la solución y es por ello que se repite su ejecución hasta que sí se encuentra solución.

A continuación se muestra la salida del programa:

```
<terminated> NQueensMinConflictApp [Java Application] /
{Reina en columna [0]=3, Reina en columna [1]=5,

Time to solve = 0.004segundos

-----Q--
---Q----
------Q-
Q-------
-------Q
-Q------
----Q---
--Q-----
```