UNIVERSIDAD DE GRANADA

Técnicas de los Sistemas Inteligentes
grado en ingeniería informática

Práctica 3

Planificación Clásica (PDDL)

Autor
José María Sánchez Guerrero

Rama
Computación y Sistemas Inteligentes || Grupo 2 - Pablo Mesejo

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

Curso 2019-2020

Índice

Introducción

En esta práctica tendremos que realizar varias tareas en el mundo de *StarCraft*, confeccionando un dominio de planificación clásico mediante el lenguaje PDDL. Estos ficheros se ejecutarán en el planificador *Metric-FF* que nos sacará un plan válido.

Se dispondrá de un par de ficheros por cada ejercicio (dominio y problema), los cuales ejecutaremos con la siguiente orden:

```
./ff -o <ruta-dominio>-f <ruta-problema>-s 0
```

El código para cada uno de los ejercicios es incremental con respecto al anterior, es decir, voy metiendo pequeñas modificaciones dependiendo de lo que se nos pida. Esto no sucede en el último ejercicio, ya que he tenido que cambiar varias cosas para que se ejecute en un tiempo razonable. Los cambios los explicaré más detalladamente después.

Ejercicio 1

En este ejercicio se nos pide que implementemos todo el dominio de *StarCraft*, es decir, edificios, unidades, recursos y acciones básicas como moverse, asignar o construir edificios.

La declaración de los tipos, las constantes y predicados no tiene mucho misterio, asi que viendo el código puede verse fácilmente cómo y por qué están declaradas. En cuanto a las acciones si las vamos a explicar mejor:

- **Navegar.** Mueve una unidad entre una localización y otra. Para ello, primero comprueba si la localización en la que se encuentra está conectada con la localización a la que va; y despues se asegura que la unidad no estaba extrayendo ningún recurso. Si esto se cumple, añade la unidad a la nueva localización y la elimina de la anterior.
- **Asignar.** Pone a un VCE a extraer recursos de un nodo. Si una unidad del tipo VCE se encuentra en la posición del recurso a extraer y no lo estaba extrayendo previamente, comienza a extraerlo.
- **Construir.** Ordena a un trabajador libre que construya un edificio. Al igual que antes, este trabajador tiene que ser un VCE, estar en la localización donde se va a construir y no puede estar ocupado extrayendo. Además, tenemos

que comprobar que no hay ningún edificio ya construido en esa localización, y para ello hemos utilizado el predicado '`exists`', que busca en el resto de localizaciones ese edificio. Por último, también tiene que existir otra unidad distinta, extrayendo el recurso que necesita el edifico para poder construirse.

Por otro lado, tenemos el fichero problema, en el cual declaramos y colocamos en el mapa tanto las unidades, como los edificios, como los minerales. Como el mundo está representado en forma de cuadrícula, también tendremos que definirla en el problema (en mi caso la hice gracias a la herramienta del editor online).

Ejercicio 2

En este ejercicio tenemos que modificar la acción *Asignar* para que en un recurso de gas haya que construir un **Extractor** antes de asignarlo.

Para ello, tenemos que añadir una precondición más para comprobar el recurso asignado en la localización. Una vez hecho esto, he utilizado un efecto condicional para que se asigne al recurso, bien si en la precondición no era un recurso de gas, o bien, si era un recurso de gas pero tiene un extractor construido.

El ejercicio nos pide construir un barracón, el cual no necesita gas, asi que para probarlo he puesto que en vez de minerales necesite estar extrayendo gas.

Ejercicio 3

Tal y como habíamos definido la acción de **Construir** en los ejercicios anterior, nos damos cuenta de que con un único VCE asignado a un recurso era suficiente para que lo construyese. Eso es lo que vamos a solucionar ahora, es decir, vamos a tener en cuenta que un edificio puede requerir de más de un tipo de recurso.

Hemos añadido el predicado '*generando ?recurso*', que se añade cuando un VCE se asigna en uno. Con esto podemos controlar más fácilmente los recursos que se están generando o no. Después, en la propia acción de construir, en vez de comprobar si hay alguien extrayendo o no, hemos utilizado el predicado '*forall*' para recorrer todos los recursos existentes y comprobar si el edificio a construir no lo necesita, o si lo necesita pero se está generando.

Por ahora, no vamos a necesitar ninguna función que elimine los predicados de generar ya que tampoco hay ninguna que desasigne a las unidades de la extracción

de recursos. Para comprobar que funciona, he puesto que también se construya el centro de mando, ya que es el único que necesita de todos los recursos.

Ejercicio 4

Ejercicio 5

Ejercicio 6