Universidad Carlos III

Ingeniería del Conocimiento

Curso 2021-22

Tutorial 2:

PDDL: Dominios de Planificación

Ana Tian Villanueva Conde (100405817)

Yolanda Luque Hazañas (100405948)

Estudiar los dominios y problemas de planificación proporcionados en el directorio domains. Explicar en la memoria (1) que hace cada dominio a alto nivel y (2) detallar que representan los predicados y la acciones con sus precondiciones y efectos.

Para estudiar los dominios hemos escogido el dominio de “blocks” y hemos corrido ./plan ../domains/blocks/domain.pddl ../domains/blocks/probBLOCKS-4-0.pddl blocks/salida.

Primero vamos a explicar el funcionamiento del problema de bloques de manera conceptual y a continuación explicaremos con más detalle el código utilizado.

El problema consiste en varias pilas de bloques numerados que se superponen uno encima de otro. El objetivo es mover los bloques de una pila a otra, colocando los bloques uno encima de otro. Para desplazar los bloques tengo que utilizar una mano robot (gripper) que realice las acciones de levantar el bloque, desplazar el bloque y soltar el bloque, apilar y desapilar. Para modelar el problema vamos a nombrar los bloques: A, B, C, D.

Comenzamos definiendo el problema utilizando (define (domain BLOCKS) y los requirements necesarios para su ejecución que son strips (adds y deletes) y typing (para definir tipos, en este caso block).

Los predicados que utilizaremos son:

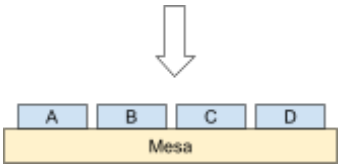
- (on ?x ?y): para indicar que un bloque x está encima de otro bloque y.
- (ontable ?x): para indicar que un bloque está encima de la mesa.
- (clear ?x): para indicar que encima de nuestro bloque x no se encuentra otro bloque.
- (handempty): para indicar que el brazo del robot (gripper) está vacío.
- (holding ?x): indica que el gripper está ocupado por el bloque x.

Las acciones que utilizamos son:

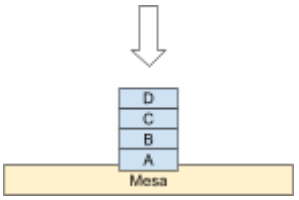
- **pickup**: cuyo parámetros es el bloque ?x. Las precondiciones para que se active esta regla son que el gripper esté vacío (handempty), que el bloque esté en la mesa (no en una pila) y que encima del bloque que queremos coger no se encuentre otro bloque (clear). Los efectos son las consecuencias después de realizar la acción, que en este caso son que el bloque que hemos cogido ya no se encuentre en la mesa (pues ahora está elevado por la mano del robot, el robot ya no está vacío y por tanto está sujetando el bloque y que encima del bloque ya no se encuentre nada (pues está la mano del robot).
- **put-down**: se trata de la acción contraria que anteriormente, pues el robot comienza con las precondiciones (holding ?x), pues está sujetando el bloque y termina con el efecto de que la mano queda libre (suelta el bloque) y el bloque se encuentra ahora en la mesa (ontable ?x).
- **stack**: con dos parámetros (bloque ?x ?y), donde x e y son distintos bloques. Esta acción consiste en colocar un bloque x encima de un bloque y. Para ello, la mano del robot tiene que tener el bloque (holding ?x) y el bloque y, donde el robot quiere colocar el bloque x que está sujetando está libre por encima (clear ?x). Los efectos son que ya no sujeta el bloque (pues lo ha colocado encima de otro bloque), ahora ya el bloque y no está clear, pero el bloque x si lo está (pues encima del bloque y se ha colocado el bloque x).
- **unstack**: al igual que antes tiene dos parámetros, el bloque que quiere soltar (x) y el bloque sobre el que quiere soltar (y). Las precondiciones son que un bloque x se encuentre sobre otro bloque y, que la mano del robot esté vacía y que el bloque y no tenga encima otro bloque. Los efectos son que la mano del robot está ocupada (holding ?x), el bloque x ya no está sobre el bloque y, y que x ya no está clear.

El primer ejemplo que hemos ejecutado es el de bloques 4-0, que comienza con un estado inicial en el que los 4 bloques se encuentran situados encima de la mesa y por tanto ninguno de los bloques está apilado. El objetivo final es colocar los bloques en una única pila de tal manera que A esté sobre la mesa, B sobre A, C sobre B y D sobre C como mostramos en la imagen.

Initial state:



Goal state:



Los resultados obtenidos son los siguientes:

NActions: 6 Time: 0.00 Cost: 6.00 0: (PICK-UP B) 1: (STACK B A) 2: (PICK-UP C) 3: (STACK C B) 4: (PICK-UP D) 5: (STACK D C)	NActions: 10 Time: 0.00 Cost: 10.00 0: (PICK-UP D) 1: (STACK D C) 2: (PICK-UP B) 3: (STACK B A) 4: (UNSTACK D C) 5: (PUT-DOWN D) 6: (PICK-UP C) 7: (STACK C B) 8: (PICK-UP D) 9: (STACK D C)

Podemos observar que este problema tiene dos soluciones, una más corta, con un coste de 6, pues tan solo realiza 6 acciones y otra más larga, con un coste de 10.

A continuación realizamos otra prueba del mismo problema pero con diferentes condiciones iniciales. En este caso hemos decidido implementar el programa con 12 bloques y el tiempo de ejecución es considerablemente más alto y no hemos conseguido que termine la ejecución.

Decidimos probar con un número menor que 12 bloques para poder obtener un ejemplo más complejo que el de los 4 bloques y probamos con 8, con el cual obtuvimos una solución con 34 acciones y un tiempo de ejecución de 3.34 segundos.

Fijaros en los dominios Satellite y Zenotravel y comparar cómo se modela en cada uno la cantidad de fuel utilizada. Explicar en la memoria las diferencias.

La principal diferencia entre Satellite y Zenotravel es que Satellite importa los requirements de typing, fluents y equality para modelar el problema, mientras que Zenotravel utiliza únicamente typing.

Typing nos importa utilizar tipos (types) en la declaración de las variables. En el caso de Satellite utiliza satellites, direction, instrument y mode, mientras que Zenotravel utiliza aircraft, person, city, flevel como objetos.

El modelo de Satellite utiliza valores numéricos gracias a la etiqueta de (:fluents) y por tanto hacer uso de las (:functions …) para comprobar que se cumplen los valores. Por tanto, en este problema podemos utilizar operadores como igual, mayor y menor (importando :equality). Por el contrario, el modelo de Zenotravel no admite estos valores numéricos y por tanto tiene que modelar el problema con tan solo valores booleanos.

El resultado obtenido tras la ejecución de Satellite es un único resultado con 11 acciones pero un coste de 108.59, bastante mayor que el coste obtenido por Zentrotravel. Este último obtiene 3 salidas que llevan a la misma solución pero con diferente coste. La primera salida obtenida tiene un coste de 13, la segunda salida tiene un coste de 12, mientras que la última salida el coste es de 11.

Comparación de los mejores resultados de Satellite (izquierda) y Zenotravel (derecha).

```
;NActions: 11
;Time:      0.00
;Cost:  108.59
   0: (SWITCH_ON INSTRUMENT0 SATELLITE0)
   1: (TURN_TO SATELLITE0 PHENOMENON4
PHENOMENON6)
   2: (TURN_TO SATELLITE0 GROUNDSTATION2
PHENOMENON4)
   3: (CALIBRATE SATELLITE0 INSTRUMENT0
GROUNDSTATION2)
   4: (TURN_TO SATELLITE0 PHENOMENON4
GROUNDSTATION2)
   5: (TAKE_IMAGE SATELLITE0 PHENOMENON4
INSTRUMENT0 THERMOGRAPH0)
   6: (TURN_TO SATELLITE0 PHENOMENON6
PHENOMENON4)
   7: (TAKE_IMAGE SATELLITE0 PHENOMENON6
INSTRUMENT0 THERMOGRAPH0)
   8: (TURN_TO SATELLITE0 PHENOMENON3
PHENOMENON6)
   9: (TURN_TO SATELLITE0 STAR5 PHENOMENON3)
  10: (TAKE_IMAGE SATELLITE0 STAR5 INSTRUMENT0
THERMOGRAPH0)
```

```
;NActions: 11
;Time:   0.00
;Cost:   11.00
   0: (BOARD PERSON4 PLANE1 CITY1)
   1: (ZOOM PLANE1 CITY1 CITY0 FL6 FL5
FL4)
   2: (BOARD PERSON2 PLANE1 CITY0)
   3: (BOARD PERSON3 PLANE1 CITY0)
   4: (ZOOM PLANE1 CITY0 CITY3 FL4 FL3
FL2)
   5: (BOARD PERSON1 PLANE1 CITY3)
   6: (DEBARK PERSON2 PLANE1 CITY3)
   7: (DEBARK PERSON3 PLANE1 CITY3)
   8: (DEBARK PERSON4 PLANE1 CITY3)
   9: (ZOOM PLANE1 CITY3 CITY2 FL2 FL1
FL0)
  10: (DEBARK PERSON1 PLANE1 CITY2)
```

Experimentos: Por cada dominio hacer las siguientes acciones:
a) Limitar el tiempo de ejecución de cada problema a 2 minutos. Para ello, en el terminal desde donde se invoque al planificador, ejecutar al principio (con una vez es suficiente) el comando `ulimit -t 120`.
b) Ejecutar el planificador para resolver los 5 problemas proporcionados en cada directorio y analizar las soluciones proporcionadas. Los problemas del subdirectorio `otros` completan los utilizados en las IPC, no hace falta intentar resolverlos todos, solo los 5 del directorio padre.
c) Escribir en la memoria una tabla con el tiempo que tarda en solucionar cada problema, el número de soluciones generadas y por cada solución la longitud del plan y la calidad del plan (si en el problema no se define ninguna métrica se asume que es la longitud del plan).
d) Escribir en la memoria el número de objetos y metas que tiene cada problema.
e) Definir un problema nuevo (que no sea trivial) que el planificador sea capaz de resolver. Escribir en la memoria el número de objetos y metas que tiene y los resultados de la ejecución tal como se piden en el apartado c).

Blocks	Logistics	Rovers
NActions: 6 Time: 0.00 Cost: 6.00 Pruebas: 2 (STACK D C)	NActions: 20 Time: 0.00 Cost: 20.00 Pruebas: 2 (UNLOAD-TRUCK OBJ21 TRU1 POS1)	NActions: 10 Time: 0.00 Cost: 0.00 Pruebas: 1 (COMMUNICATE_IMAGE_DATA ROVER0 GENERAL OBJECTIVE1 HIGH_RES WAYPOINT2 WAYPOINT0)
NActions: 34 Time: 3.34 Cost: 34.00 Pruebas: 1 (STACK D F)	NActions: 36 Time: 0.00 Cost: 36.00 Pruebas: 2 (UNLOAD-AIRPLANE OBJ13 APN1 APT3)	NActions: 43 Time: 0.01 Cost: 0.00 Pruebas: 2 (COMMUNICATE_ROCK_DATA ROVER3 GENERAL WAYPOINT0 WAYPOINT4 WAYPOINT1)
---	NActions: 70 Time: 6.22 Cost: 70.00 Pruebas: 9 (UNLOAD-AIRPLANE OBJ31 APN1 APT4)	NActions: 57 Time: 6.88 Cost: 5.00 Pruebas: 2 (COMMUNICATE_IMAGE_DATA ROVER0 GENERAL OBJECTIVE2 HIGH_RES WAYPOINT0 WAYPOINT2)
---	NActions: 79 Time: 4.57 Cost: 79.00 Prueba: 6 (UNLOAD-TRUCK OBJ52 TRU1 POS1)	NActions: 47 Time: 12.11 Cost: 3.00 Pruebas: 3 (COMMUNICATE_SOIL_DATA ROVER2 GENERAL WAYPOINT8 WAYPOINT8 WAYPOINT9)

---	NActions: 73 Time: 5.05 Cost: 73.00 Pruebas: 6 (UNLOAD-AIRPLANE OBJ21 APN1 APT4)	NActions: 84 Time: 24.89 Cost: 7.00 Pruebas: 2 (COMMUNICATE_IMAGE_DATA ROVER3 GENERAL OBJECTIVE5 COLOUR WAYPOINT0 WAYPOINT6)

Satellite	**Zenotravel**
NActions: 11 Time: 0.00 Cost: 108.59 Prueba: 1 (TAKE_IMAGE SATELLITE0 STAR5 INSTRUMENT0 THERMOGRAPH0)	Nactions: 12 Time: 0 Cost: 12 Pruebas: 3 (DEBARK PERSON1 PLANE1 CITY2)
NActions: 18 Time: 0.03 Cost: 93.50 Pruebas: 2 (TURN_TO SATELLITE1 GROUNDSTATION2 PLANET9)	NActions: 27 Time: 3.12 Cost: 27.00 Pruebas: 6 (DEBARK PERSON8 PLANE2 CITY3)
NActions: 73 Time: 0.35 Cost: 158.02 Pruebas: 2 (TAKE_IMAGE SATELLITE2 STAR15 INSTRUMENT4 SPECTROGRAPH1)	NActions: 48 Time: 2.86 Cost: 48.00 Pruebas: 4 (DEBARK PERSON15 PLANE1 CITY6)
NActions: 178 Time: 40.01 Cost: 325.30 Pruebas: 4 (TURN_TO SATELLITE2 STAR13 PLANET7)	NActions: 80 Time: 7.21 Cost: 80.00 Pruebas: 5 (DEBARK PERSON20 PLANE2 CITY13)
NActions: 74 Time: 6.23 Cost: 67.19 Pruebas: 4 (TAKE_IMAGE SATELLITE1 STAR10 INSTRUMENT6 IMAGE1)	Nactions: 124 Time: 100.89 Cost: 124 Pruebas: 3 (DEBARK PERSON19 PLANE3 CITY16)

Para realizar el nuevo problema hemos decidido realizar el siguiente modelo:

Conceptos: 9 fichas (tiles) y 9 celdas (cells). Las fichas se encuentran colocadas, en un principio aleatoriamente, excepto una celda que está vacía. El objetivo es ordenar las fichas del 1 al 8 en las celdas del 1 al 8 correspondientes, excepto la última celda que está vacía. Para poder realizar este movimiento, la celda a la que se quiere mover ha de estar vacía.

Predicados:

- on(x, y) la ficha x está en la celda y.
- clear(y) la celda y está vacía.
- adjacent(y, z) la celda y tiene una celda adyacente z.

Estado inicial: (adjacent c1 c2) (adjacent c2 c3) (adjacent c4 c5) (adjacent c5 c6) (adjacent c7 c8) (adjacent c8 c9) (on t5 c1) (on t2 c2) (on t3 c6) (on t8 c4) (clear c5) (on t4 c6) (on t7 c7) (on t1 c8) (on t6 c9).

Estado final: las celdas ordenadas es su celda correspondiente (on t1 c1) (on t2 c2) (on t3 c3) (on t4 c4) (on t5 c5) (on t6 c6) (on t7 c7) (on t8 c8) (clear c9)).

Acciones:

- move(x, y, z): donde x es la ficha, y es la posición en la que se encuentra y z es la celda a la que quiere desplazarse la ficha.