Técnicas de los Sistemas Inteligentes (2015-2016)
Grado en Ingeniería Informática
Universidad de Granada

Práctica 3: Segunda Parte

Francisco Carrillo Pérez

4 de julio de 2016

Índice

Índice de figuras

Introducción

En el siguiente documento se va a proceder a la explicación a las decisiones de diseño que se han tomado a la hora de realizar los 4 ejercicios planteados.

1. Ejercicio 1

El problema del dominio es que no hay nada descrito para que si el avión y un pasajero no se encuentran en una misma ciudad, el avión viaje el solo a esas ciudad donde se encuentra el pasajero. Se ha modificado el dominio añadiendo un tercer :method, como se puede observar a continuación.

```
;se añade este nuevo caso en el que ni el avión ni la persona están en la misma ciudad
; en este caso que vuele el avion a la ciudad donde está la persona, que la persona coja
; el vuelo y le deje en la ciudad que se desea.
(:method Case3
    :precondition (and (at ?p - person ?c1 - city)
                       (at ?a - aircraft ?c2 - city) (different ?c1 - city ?c2 - city))

    :tasks (
                (fly ?a ?c2 ?c1)
                (board ?p ?a ?c1)
                (mover-avion ?a ?c1 ?c)
                (debark ?p ?a ?c )))
)
```

De esta forma, si el avión y una persona no está en una misma ciudad, el avión se desplazará.
Como precondiciones tenemos que las ciudades en las que se encuentras la persona y el avión sean distintas. Las tareas que se realizan son:

- El avión vuela a la ciudad donde está la persona.
- La persona embarca.
- El avión vuelva a la ciudad a donde queremos desplazar a la persona.
- La persona desembarca en la ciudad.

2. Ejercicio 2

En este problema tenemos una restricción en referente al fuel. Esto puede ser debido a que no tenemos suficiente fuel para hacer todos los viajes, por lo que tendremos que hacer acciones de repostaje. Por lo tanto deberémos implementar un método para repostar el fuel. Se puede observar a continuación mi implementación:

```
(:method no-hay-suficiente-fuel
;; este método se escogerá para usar la acción fly siempre que el avión no
;;tenga fuel para volar desde ?c1 a ?c2
;; si no hay fuel suficiente el método se aplicará y repondrá fuel para poder volar
     :precondition (not (hay-fuel ?a ?c1 ?c2))
     :tasks (
                 (refuel ?a ?c1)
                 (fly ?a ?c1 ?c2)
           )
)
```

Si no hay fuel, entonces primero repondrá fuel, y a continuación volará a la ciudad.

3. Ejercicio 3

Ahora, con respecto al ejercicio 2, deseamos tener un límite de fuel consumido y que se prioricen las acciones de volado rápido sobre las lentas.

1. He añadido dos predicados *hay-fuel-rapido* y *hay-fuel-lento* para hacerlos derivados y se pueda volar sin necesidad de respostar.

   ```
   (hay-fuel-rapido ?a ?c1 ?c2)
   (hay-fuel-lento ?a ?c1 ?c2)
   ```

2. He modificado la tarea de mover-avion para que priorice la forma de volar rápida que la forma de volar lenta. Además se pone como precondición que no puede sobrepasar el límite de fuel que se ha indicado.

   ```
   (:task mover-avion
     :parameters (?a - aircraft ?c1 - city ?c2 -city)
     (:method suficiente-fuel-rapido
     :precondition(and(hay-fuel-rapido ?a ?c1 ?c2)
     (>(fuel-limit)(total-fuel-used)))
         :tasks (
     (zoom ?a ?c1 ?c2)
     )
     )

     (:method no-suficiente-fuel-rapido
     :precondition (and (not (hay-fuel-rapido ?a ?c1 ?c2))
     (> (fuel-limit) (total-fuel-used)) )
     :tasks (
      (refuel ?a ?c1)
      (zoom ?a ?c1 ?c2)
   ```

```
   )
  )

  (:method suficiente-fuel-lento
   :precondition (and (hay-fuel-lento ?a ?c1 ?c2)
   (> (fuel-limit) (total-fuel-used)))
   :tasks (
   (fly ?a ?c1 ?c2)
   )
  )

  (:method no-suficiente-fuel-lento
  (> (fuel-limit) (total-fuel-used)) )
   :tasks (
   (refuel ?a ?c1)
   (fly ?a ?c1 ?c2)
   )
  )
)
```

4. Ejercicio 4

Para este ejercicio se han realizado distintas cosas:

1. Para representar que hay varios pasajeros se han añadido los siguientes predicados:

   ```
   (unpasajero)
   (maxpasajeros ?a - aircraft)
   (pasajeros ?a - aircraft)
   (destino ?p - person ?c - city)
   ```

   Un ejemplo de añadirlo en la descripción del problema sería, que el máximo de pasajeros va a ser 4 para el avión 1, que hay actualmente 0 pasajeros, y que la unidad de pasajeros para incrementar o decrementar será 1. También se ha añadido el destino de cada pasajero, pero debe ser el mismo que se pone en el goal.

   ```
   (= (maxpasajeros a1) 4)
   (= (pasajeros a1) 0)
   (= (unpasajero) 1)
   (destino p1 c5)
   (destino p2 c3)
   (destino p3 c4)
   ```

2. Se han modificado las precondiciones en los métodos de la tarea mover avión de la siguiente forma:

```
;Si hay sificiente fuel rápido
(and (> (fuel ?a) (* (distance ?c1 ?c2) (fast-burn ?a)))
(> (fuel-limit) (total-fuel-used)))
;Si no hay suficiente fuel rápido
(and (not (> (fuel ?a) (* (distance ?c1 ?c2) (fast-burn ?a))))
(> (fuel-limit) (total-fuel-used)) )
;Si hay suficiente fuel lento
(and (> (fuel ?a) (* (distance ?c1 ?c2) (slow-burn ?a)))
(> (fuel-limit) (total-fuel-used)))
;Si no hay suficiente fuel lento
(and (not (> (fuel ?a) (* (distance ?c1 ?c2) (slow-burn ?a))))
(> (fuel-limit) (total-fuel-used)) )
```

3. Las tardeas board y debark se han comvertido en recursivas para que se pueda embarcar y desembarcar respectivamente a varios pasajeros en una misma ciudad. Quedaría de la siguiente forma:

```
(:task board
:parameters ()
(:method recurre1
:precondition (and  (at ?p ?c)
(at ?a ?c)
(< (pasajeros ?a) (maxpasajeros ?a))
(not (destino ?p ?c))
)
:tasks  (
(boardp ?p ?a ?c)
(board)
)
)
(:method caso-base
:precondition():tasks())
)

(:durative-action boardp
:parameters (?p - person ?a - aircraft ?c - city)
:duration (= ?duration (boarding-time))
:condition (and  (at ?p ?c)
(at ?a ?c)
(< (pasajeros ?a) (maxpasajeros ?a)))
:effect (and  (not (at ?p ?c))
(in ?p ?a)
(increase (pasajeros ?a) (unpasajero))))
```

```
(:task debark
:parameters ()
(:method recurre1
:precondition (and (in ?p ?a)
(at ?a ?c)
)
:tasks  (
(debarkp ?p ?a ?c)
(debark)
)
)
(:method caso-base
:precondition():tasks())
)

(:durative-action debarkp
:parameters (?p - person ?a - aircraft ?c - city)
:duration (= ?duration (debarking-time))
:condition (and (in ?p ?a)
(at ?a ?c))
:effect (and  (not (in ?p ?a))
(at ?p ?c)
(decrease (pasajeros ?a) (unpasajero)) ))
(:task fly
:parameters (?a - aircraft ?c1 - city ?c2 -city)
(:method unico
:precondition(bind ?cost (+ (total-fuel-used)(* (distance ?c1 ?c2)(slow-burn ?a))))
:tasks  (
(flyp ?a ?c1 ?c2 ?cost)
)
)
)
```

4. No se ha conseguido realizar que se deriven unos predicados a partir de otros.

5. Se ha realizado otro problema, en este caso se han añadido unas cuantas ciudades, junto con 7 personas repartidas por distintas ciudades y dos aviones, haciendo que todas las personas tengan que acabar en Barcelona. Además de la restricción de que solo se pueden subir dos personas a un avión. Se encuentra en el fichero *problema-zeno-v05.pddl*.