Técnicas de los Sistemas Inteligentes (2018-2019)
Grado en Ingeniería Informática
Universidad de Granada

Práctica 3

UNIVERSIDAD
DE GRANADA

Antonio Jesús Heredia Castillo

8 de junio de 2019

1. Problema 1

Este es el problema mas fácil de resolver. Se trata de transportar a tres personas, de una ciudad a otra. Todos los pasajeros se encuentran en ubicaciones diferentes pero todos quieren ir a la misma ciudad. No existe ninguna restricción en este ejercicio, ni de fuel ni de tiempo.
He tenido que cambiar muy pocas cosas en este ejercicio. Simplemente he añadido un caso mas al fichero de dominio que nos daban de base. He añadido un nuevo método en la tarea de mover persona.

El nuevo caso contempla que el avión y la persona no esten en la misma ciudad. Haciendo en ese caso a que el avión se cambie a la ciudad en la que esta el pasajero, se suba, lo lleve a la ciudad destino y se baje del avión.

```
(:method Case3; si no esta en la ciudad destino y el avion no esta en la
    misma ciudad
        :precondition (and (at ?p - person ?c1 - city)
                                        (at ?a - aircraft ?c2 - city))

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

2. Problema 2

En este problema añadimos restricciones de fuel. Al principio el fuel tendra 200 de fuel y la capacidad de fuel total es de 300. El avión podrá repostar todas las veces que quiera y en las ciudades que quiera. Por lo tanto simplemente tendremos que añadir la nueva acción de repostar fuel cuando no quede.

Primero cambiamos el "derived" de hay-fuel para saber si va a quedar fuel para ir de una ciudad a otra. Como ahora mismo solo usamos la tarea "fly", solo calcularemos la quema de combustible de forma lenta.

```
  (>= (fuel ?a)  (* (distance ?c1 ?c2) (slow-burn ?a)))
```

Por otro lado añadiré un método a la tarea de "mover-avion". Este método nos servirá para cuando no haya fuel disponible para el avión. Que en cuyo caso primero rellenara el tanque del avión y luego volara a la ciudad a la que quería.

```
(:method noHayFuel
        :precondition (not(hay-fuel ?a ?c1 ?c2))
        :tasks (
                (refuel ?a  ?c1)
                (fly ?a ?c1 ?c2)

        )
)
```

3. Problema 3

A los dos problemas anteriores añadimos dos formas de "mover-avion". Habra un vuelo rapido y otro lento. Pero no gastaran lo mismo. El vuelo rápido consumirá el doble de combustible que el lento. Siempre habra que priorizad que se vaya a la maxima velocidad. Ademas habra un limite de fuel a usar, que sera de 1500 unidades.

Lo primero que haré es crear dos "derived". Uno para saber si queda fuel para ir rápido y otro para saber si queda fuel para ir lento. Estos sustituirá el derived anterior que miraba solo si habia fuel para ir lento. En estos derived nuevos se calculara el gasto de combustible en función del tipo de vuelo

```
(:derived
        (hay-fuel-rapido ?a - aircraft ?c1 - city ?c2 - city)
        (>= (fuel ?a)  (* (distance ?c1 ?c2) (fast-burn ?a)))
)

(:derived
        (hay-fuel-lento ?a - aircraft ?c1 - city ?c2 - city)
        (>= (fuel ?a)  (* (distance ?c1 ?c2) (slow-burn ?a)))
)
```

Dado esto, también tendremos que cambiar la tarea de mover el avión. Ya que ahora existe distintos tipos de vuelo y hay que priorizar el rápido. Crearemos cuatro nuevos métodos. Para si hay fuel suficiente para ir rapido y no se ha superado el limite de fuel. Otro para en caso de que no se haya superado el limite pero no queda fuel para ir a esa ciudad. Los otros dos se ejecutaran solo en caso de que se vaya a superar el limite de fuel. En ese caso el avión ira lento para gastar menos fuel. Por lo tanto cuando httd cuando intente a resolver el problema siempre intentara primero ir rápido y en caso de que no pueda empezara a ir lento para no superar el limite de fuel.

```
(:method fuel-suficiente-rapdio
        :precondition (and(hay-fuel-rapido ?a ?c1 ?c2) (< (+(total-fuel-used
            ) (* (distance ?c1 ?c2) (fast-burn ?a))) (fuel-limit) ))

        :tasks (
                (zoom ?a ?c1 ?c2)
```

```
        )
)

(:method noHayFuelRapido
        :precondition (and (not(hay-fuel-rapido ?a ?c1 ?c2)) (< (+ (total-
            fuel-used) (* (distance ?c1 ?c2) (fast-burn ?a))) (fuel-limit) )
            )

        :tasks (
                (refuel ?a  ?c1)
                (zoom ?a ?c1 ?c2)
        )
)

(:method fuel-suficiente-lento
        :precondition (and(hay-fuel-lento ?a ?c1 ?c2) (< (+(total-fuel-used)
             (* (distance ?c1 ?c2) (slow-burn ?a))) (fuel-limit) ))

        :tasks (
                (fly ?a ?c1 ?c2)
        )
)

(:method noHayFuelLento
        :precondition (and (not(hay-fuel-lento ?a ?c1 ?c2)) (< (+(total-fuel
            -used) (* (distance ?c1 ?c2) (slow-burn ?a))) (fuel-limit) ))

        :tasks (
                (refuel ?a  ?c1)
                (fly ?a ?c1 ?c2)

        )
)
```

4. Problema 4

En este problema tendremos que añadir varias características nuevas a nuestro dominio.

Primero habrá que permitir al avión tener una capacidad máxima de pasajeros que cogeran en un avión. Y por lo tanto habra que modificar las acciones de board y debark. Para esto ademas tendremos la posibilidad de subir varias personas que estén en la misma ciudad o que se bajen varias personas que ha llegado a su destino, cosas que antes no podiamos hacer. Para ello haremos uso de la recursividad, solo tendremos un task-goal que se ejecutara hasta que todo el mundo este en su ciudad. y cambiaremos los que teníamos antes por predicados del estilo ($destino?x - person?y - city$). Y por ultimo ademas de haber fuel limitado, tendremos tiempo limitado.

Como he dicho anteriormente solo existira el task-goal "ìniciar" sistema. Este task-goal se encargara de llevar a todo el mundo a su destino, este se ejecu-

tara de forma recursiva. El primer metodo es que todo el mundo este en su destino y acabara con toda la ejecución.

```
(:method TodosEnDestino
        :precondition(and
        (not ( destino ?p - person ?c - city))
        )
        :tasks()
)
```

Cuando una persona o varias, esta ya en su destino , eliminara el predicado que indicaba que una persona tenia que ir a ese destino y vuelve hacer la llamada a esa tarea.

```
(:method YaEnDestino
        :precondition (and
        (destino ?p1 - person ?c1 - city)
        (at ?p1 ?c1)
        )
        :tasks (
                (eliminarDestino ?p1 ?c1)
                (iniciarSistema)
        )
)
```

El siguiente método sera desembarcar a una persona que ha llegado a su destino. Las pregundicion es que la persona este en un avión esta en la ciudad a la que quiere ir.

```
(:method Desembarcar ;la gente ha embarcado en el avion
:precondition (and
        (in ?p - person ?a - aircraft)
        (at ?a - aircraft ?c1 - city)
        (destino ?p ?c1)
        )

        :tasks (
                (debark ?p ?a ?c1 )
                (iniciarSistema)
        )
)
```

La siguiente tarea sera si hay un avión en el mismo sitio donde esta una persona que quiere ir a alguna ciudad, esta embarca en el avión. Esta acción se realiza antes de la que volar para que en caso de que haya varias personas esperando en la ciudad todas se puedan subir al avión.

```
(:method Embarcar ;si no esta en la ciudad destino, pero avion y persona
    estan en la misma ciudad
        :precondition (and
        (at ?p - person ?c1 - city)
        (at ?a - aircraft ?c1 - city)
        (destino ?p - person ?ciudad - city)
        )

        :tasks (
```

```
                (board ?p ?a ?c1)
                (iniciarSistema)
        )
)
```

La siguiente es que una vez que todo el mundo esta embarcado, el avión pueda cambiar de ciudad a donde quiere ir la ultima persona que se ha embarcado.

```
(:method Volar ;la gente ha embarcado en el avion
        :precondition (and
        (at ?a - aircraft ?c1 - city)
        (in ?p ?a)
        (destino ?p - person ?c2 - city)
        )

        :tasks (
                (mover-avion ?a ?c1 ?c2)
                (iniciarSistema)
        )
)
```

Y por ultimo tendremos la acción de llevar el avión a una zona donde hay gente esperando.

```
(:method LlevarAvion
        :precondition (and (at ?p - person ?c1 - city)
        (at ?a - aircraft ?c2 - city)
        (destino ?p - person ?c3 - city)
        )

        :tasks (
                (mover-avion ?a ?c2 ?c1)
                (iniciarSistema)
        )
)
```

Con esto conseguimos la capacidad de que varia gente se suba al avión. Aunque las acciones de embarcar y desembarcar son las que llevan el control del limite de gente.
Si puede embarcar una persona mas lo hara y sumara uno al contador de gente en el avión.

```
(:durative-action board
        :parameters (?p - person ?a - aircraft ?c - city)
        :duration (= ?duration (boarding-time))
        :condition (and  (at ?p ?c)
                (at ?a ?c)
                (> (capacidad ?a) (cantPasajeros ?a))
        )
        :effect (and  (not (at ?p ?c))
                (in ?p ?a)
                (increase (cantPasajeros ?a) 1)
        )
)
```

El acto de desembarcar es igual que el de embarcar pero sin comprobar la capacidad del avión y restando a uno a la cantidad de gente en el avión. También he cambiado las acciones de volar rapido y lento para incrementar el tiempo gastado para poder llevar a cabo el control del limite de tiempo que podemos establecer.

```
:effect (and  (not (at ?a ?c1))
        (at ?a ?c2)
        (increase (total-fuel-used)
        (* (distance ?c1 ?c2) (slow-burn ?a)))
        (increase (fuel-used ?a)
        (* (distance ?c1 ?c2) (slow-burn ?a)))
        (increase
        (tiempo ?a) (/ (distance ?c1 ?c2) (slow-speed ?a))
        )
        (decrease (fuel ?a)
        (* (distance ?c1 ?c2) (slow-burn ?a))
)
```