

Técnicas de los Sistemas Inteligentes (2014-2015)

Grado en Ingeniería Informática

Universidad de Granada

Relación de Ejercicios PDDL

M^a Cristina Heredia Gómez

15 de junio de 2015

Índice

1. Escribir un dominio de planificación en PDDL para que un planificador pueda encontrar planes de actuación para uno o varios robots como soluciones a problemas de distribución de paquetes entre habitaciones. En el material de esta sesión de prácticas hay un fichero ejemplo de un problema para este tipo de dominio, que se corresponde con el estado inicial y objetivo en la figura de más abajo. Probar que el dominio escrito genera plan para distintos problemas (definidos por el alumno) en los que varíe la cantidad de robots, la cantidad de paquetes a distribuir y la cantidad de habitaciones conectadas. 3
2. Escribir un dominio de planificación en PDDL, modificando el dominio del anterior ejercicio, de tal manera que se tenga en cuenta que la acción de moverse de una habitación a otra consume una cantidad de batería y, por tanto, requiere que el robot tenga nivel de batería para moverse. Además, considerar que hay una nueva acción de carga de batería que permite reponer la batería. Considerar para ello que se ha definido un predicado (cambio $n1$ $n2$ – nivelbat) que representa un cambio en el nivel de batería desde un nivel $n1$ a un nivel $n2$. En el material de esta sesión de prácticas hay un fichero ejemplo de un problema para este tipo de dominio. 7
3. Escribir un dominio de planificación en PDDL, modificando el dominio del anterior ejercicio, de manera que se puedan utilizar ahora dos acciones diferentes, moverse rápido y moverse lento tales que moverse rápido consume más unidades de fuel que moverse lento. Probarlo con varios problemas. 11
4. Escribir un dominio de planificación en PDDL 2.1 que responda a los requisitos explicados en los anteriores ejercicios, utilizando las capacidades expresivas de PDDL 2.1, es decir, representando funciones numéricas. Al igual que en los anteriores ejercicios, definir distintos problemas para comprobar que vuestro dominio es correcto. 15

Índice de figuras

1.1. descripción gráfica del problema, con estados final e inicial	3
1.2. descripción en código del problema	4
1.3. descripción del dominio para problema Ejercicio1	5
1.4. plan calculado para problema Ejercicio1	6
1.5. plan calculado para problema Ejercicio1 modificado añadiendo 3 elementos	7
2.1. problemaEjercicio2.pddl, con 3 niveles de batería, 1 robot, 2 paquetes y 3 habitaciones	8
2.2. Dominio para problema Ejercicio2	9
2.3. plan calculado para problema Ejercicio2	10
2.4. plan calculado para problema Ejercicio2	10
3.1. Dominio solución para problema Ejercicio3	11

3.2. descripción problemaEjercicio3.pddl	13
3.3. plan calculado para problema Ejercicio3	13
3.4. plan calculado para problema Ejercicio3 con 2 robots,4 habitaciones,3 paquetes y 2 veloci- dades de movimiento	14
4.1. dominio solución propuesto al ejercicio4	15
4.2. descripción problemaEjercicio4.pddl	17
4.3. plan solución propuesto al problemaEjercicio4	18
4.4. plan solución propuesto al problemaEjercicio4mod.pddl	18

1. Escribir un dominio de planificación en PDDL para que un planificador pueda encontrar planes de actuación para uno o varios robots como soluciones a problemas de distribución de paquetes entre habitaciones. En el material de esta sesión de prácticas hay un fichero ejemplo de un problema para este tipo de dominio, que se corresponde con el estado inicial y objetivo en la figura de más abajo. Probar que el dominio escrito genera plan para distintos problemas (definidos por el alumno) en los que varíe la cantidad de robots, la cantidad de paquetes a distribuir y la cantidad de habitaciones conectadas.

Inicialmente se nos da el problema *problemaEjercicio1.pddl*:

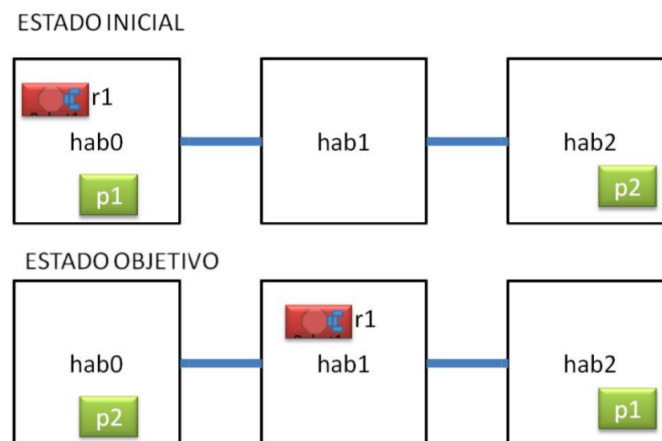


Figura 1.1: descripción gráfica del problema, con estados final e inicial

```

(define (problem RDistribuye-1)
  (:domain RobotDistribuidor)
  (:objects
    r1 - robot
    p1 - paquete
    p2 - paquete
    hab0 - habitacion
    hab1 - habitacion
    hab2 - habitacion
  )
  (:init
    (libre r1)
    (at r1 hab0)
    (at p1 hab0)
    (at p2 hab2)
    (conectada hab0 hab1)
    (conectada hab1 hab0)
    (conectada hab2 hab1)
    (conectada hab1 hab2)
  )
  (:goal (and
    (at r1 hab1)
    (at p2 hab0)
    (at p1 hab2)
  ))
)

```

Figura 1.2: descripción en código del problema

para el cual se propone como una posible solución el siguiente dominio:

```

(define (domain RobotDistribuidor)
  (:requirements :typing)
  (:types objeto - object
    paquete robot - objeto
    habitacion)

  (:predicates
    (at ?r - objeto ?h - habitacion)
    (conectada ?h1 - habitacion ?h2 - habitacion)
    (libre ?r - robot)
    (cogido ?paq - paquete ?r - robot)
  )

  (:action mover
    :parameters (?r - robot ?origen ?destino - habitacion)
    :precondition (and
      (at ?r ?origen)
      (conectada ?origen ?destino))
    :effect (and
      (at ?r ?destino)
      (not (at ?r ?origen)))
  )

  (:action coger
    :parameters (?hab - habitacion ?paq - paquete ?r - robot)
    :precondition (and
      (at ?paq ?hab)
      (libre ?r)
      (at ?r ?hab)
    )
    :effect (and
      (not (libre ?r))
      (cogido ?paq ?r)
      (not (at ?paq ?hab))
    )
  )

  (:action dejar
    :parameters (?hab - habitacion ?paq - paquete ?r - robot)
    :precondition (and
      (cogido ?paq ?r)
      (at ?r ?hab))
    :effect (and
      (at ?paq ?hab)
      (libre ?r)
      (not (cogido ?paq ?r)))
  )
)

```

Figura 1.3: descripción del dominio para problema Ejercicio1

• Explicación: Vemos que el dominio que se plantea para resolver este problema, tendremos, por una parte, los objetos robot y paquete, que heredarán directamente del tipo objeto(object), y por otra, el tipo habitación. Como **predicados** tenemos:

1. **at(robot,habitación)** que será cierto si el robot está en la habitación y falso en otro caso.
2. **conectada(habitación1,habitación2)** será cierto si las habitaciones 1 y 2 están conectadas.
3. **libre(robot)** será cierto si el robot no se encuentra llevando ningún paquete.
4. **cogido(paquete,robot)** será cierto si el paquete se encuentra cogido por el robot.

Por último, como **acciones** del mundo se describen las básicas:

1. **mover(robot,habitacionOrigen,habitacionDestino)** que tendrá como efecto, que el robot se desplaza de una habitación origen a una habitación de destino.(Siempre que este se encuentre en la habitación origen).
2. **coger(habitacion,paquete,robot)** que tendrá como efecto que el robot coge el paquete que hay en la habitación donde se encuentra. Ésto último es lógico, pues si el robot no se encuentra en

la misma habitación que el paquete, no podrá tomarlo. Además, para llevar a cabo esta acción el robot deberá estar libre(no deberá ir transportando ningún otro paquete), pues en otro caso no podría cogerlo.

3. **dejar(habitacion,paquete,robot)** que tendrá como efecto que el robot deja el paquete **que tiene cogido** en la habitación que se indica, quedando así tanto el paquete como el robot libres, y en la habitación.

Si lo ejecutamos, vemos que el planificador nos da un plan para poder alcanzar nuestro goal, dadas nuestras condiciones iniciales:

```
ff: found legal plan as follows
step  0: COGER R1 P1 HAB0
       1: MOVER R1 HAB0 HAB1
       2: MOVER R1 HAB1 HAB2
       3: DEJAR R1 P1 HAB2
       4: COGER R1 P2 HAB2
       5: MOVER R1 HAB2 HAB1
       6: MOVER R1 HAB1 HAB0
       7: DEJAR R1 P2 HAB0
       8: MOVER R1 HAB0 HAB1
```

Figura 1.4: plan calculado para problema Ejercicio1

que vemos que resuelve nuestro problema. Sin embargo, éste problema que estábamos considerando era bastante sencillo, pues teníamos un único robot, tres habitaciones y dos paquetes.

Probamos ahora con un problema algo más complejo. Añadimos para ello tres elementos más: un robot, un paquete y una habitación, como se describe en el fichero *problemaEjercicio1mod.pddl* (incluido en el .zip) y vemos que igualmente nos da un plan solución.

```

ff: found legal plan as follows

step  0: COGER HAB2 P2 R2
      1: MOVER R1 HAB0 HAB1
      2: MOVER R2 HAB2 HAB1
      3: MOVER R2 HAB1 HAB0
      4: DEJAR HAB0 P2 R2
      5: MOVER R2 HAB0 HAB1
      6: MOVER R1 HAB1 HAB0
      7: MOVER R2 HAB1 HAB0
      8: COGER HAB0 P1 R1
      9: COGER HAB0 P3 R2
     10: MOVER R2 HAB0 HAB1
     11: MOVER R1 HAB0 HAB1
     12: MOVER R1 HAB1 HAB2
     13: DEJAR HAB2 P1 R1
     14: MOVER R1 HAB2 HAB1
     15: MOVER R2 HAB1 HAB2
     16: MOVER R2 HAB2 HAB3
     17: DEJAR HAB3 P3 R2
     18: MOVER R2 HAB3 HAB2
     19: MOVER R2 HAB2 HAB1

```

Figura 1.5: plan calculado para problema Ejercicio1 modificado añadiendo 3 elementos

2. **Escribir un dominio de planificación en PDDL, modificando el dominio del anterior ejercicio, de tal manera que se tenga en cuenta que la acción de moverse de una habitación a otra consume una cantidad de batería y, por tanto, requiere que el robot tenga nivel de batería para moverse. Además, considerar que hay una nueva acción de carga de batería que permite reponer la batería. Considerar para ello que se ha definido un predicado (cambio n1 n2 – nivelbat) que representa un cambio en el nivel de batería desde un nivel n1 a un nivel n2. En En el material de esta sesión de prácticas hay un fichero ejemplo de un problema para este tipo de dominio.**

Se nos da un ejemplo de problema *problemaEjercicio2.pddl* en el que tenemos definidos **tres niveles de batería (0,1,2)** y se pide crear un dominio PDDL que resuelva el problema. Para ello, se nos pide definir un predicado (**cambio n1 n2 – nivelbat**) . El cambio de unos niveles de batería a el posterior (suponiendo que cuando llega a nivel 0 se carga), lo añadimos al problema, así como los estados iniciales de que **el robot se encuentra libre y con la batería cargada**:


```

(define (problem RDistribuye-1)
  (:domain RobotDistribuidor)
  (:objects
    r1 - robot
    p1 - paquete
    p2 - paquete
    hab0 - habitacion
    hab1 - habitacion
    hab2 - habitacion
    fl0 - flevel
    fl1 - flevel
    fl2 - flevel
  )
  (:init
    (libre r1)
    (batrestante r1 fl2) ; inicialmente tiene la batería cargada
    (at r1 hab0)
    (at p1 hab0)
    (at p2 hab2)
    (cambio fl2 fl1) ; disminuye la batería
    (cambio fl1 fl0) ; disminuye la batería
    (cambio fl0 fl2) ; se carga la batería
    (conectada hab0 hab1)
    (conectada hab1 hab0)
    (conectada hab2 hab1)
    (conectada hab1 hab2)
  )
  (:goal (and
    (at r1 hab1)
    (at p2 hab0)
    (at p1 hab2)
  ))
)

```

Figura 2.1: problemaEjercicio2.pddl ,con 3 niveles de batería,1 robot,2 paquetes y 3 habitaciones

El dominio que se propone como solución es el siguiente (fichero *d2.pddl*)

```

(define (domain RobotDistribuidor)

  (:requirements :typing)
  (:types objeto - object
    paquete robot - objeto
    habitacion flevel)

  (:predicates
    (at ?r - objeto ?h - habitacion)
    (conectada ?n1 - habitacion ?n2 - habitacion)
    (libre ?r - robot)
    (cogido ?paq - paquete ?r - robot)
    (batrestante ?r - robot ?bat - flevel)
    (cambio ?n1 ?n2 - flevel)
  )

  (:action mover
    :parameters (?r - robot ?origen ?destino - habitacion ?bat1 ?bat2 - flevel)
    :precondition (and
      (at ?r ?origen)
      (conectada ?origen ?destino)
      (batrestante ?r ?bat1)
      (cambio ?bat1 ?bat2))
    :effect (and
      (at ?r ?destino)
      (not (at ?r ?origen))
      (not (batrestante ?r ?bat1))
      (batrestante ?r ?bat2)))

  (:action coger
    :parameters (?hab - habitacion ?paq - paquete ?r - robot)
    :precondition (and
      (at ?paq ?hab)
      (libre ?r)
      (at ?r ?hab))
    :effect (and
      (not (libre ?r))
      (cogido ?paq ?r)
      (not (at ?paq ?hab))))

  (:action dejar
    :parameters (?hab - habitacion ?paq - paquete ?r - robot)
    :precondition (and
      (cogido ?paq ?r)
      (at ?r ?hab))
    :effect (and
      (at ?paq ?hab)
      (libre ?r)
      (not (cogido ?paq ?r))))

```

Figura 2.2: Dominio para problema Ejercicio2

• Explicación: Vemos que, al igual que en el apartado anterior, en el dominio que se plantea para resolver el problema, tendremos por una parte los objetos robot y paquete, que heredarán directamente del tipo objeto(object), y por otra, el tipo habitación y ahora también el tipo **flevel** que representa la batería en niveles. Como **predicados** tenemos que ahora añadimos dos nuevos a los anteriores:

1. **batrestante(robot,flevel)** que indica el nivel de batería restante (flevel) para el robot.
2. **cambio(n1,n2)** indica que el nivel de batería n1 cambia al nivel de batería n2 ,cuando esta se consume.

Por último, como **acciones** mantenemos las básicas del apartado anterior, pero modificamos **mover**:

1. **mover(robot,habitacionOrigen,habitacionDestino,bateriaActual,bateriaPosterior)** que tendrá como efecto, que el robot se desplaza de una habitación origen a una habitación de destino, siempre que el robot se encuentre en la habitación origen y que tenga batería. Como resultado de esta acción, el robot se desplazará a la siguiente habitación pero también decrementará su nivel de batería, e incluso, si ésta llegara a nivel 0 ,ésta se cargará automáticamente para que pueda llevar a cabo la acción. Esto suponiendo que, como el enunciado dice, hay una acción de carga de batería, que en nuestro caso la modelaremos indicando en el predicado **cambio** que del nivel 0 se cambia al nivel 2.

Si lo ejecutamos, vemos que el planificador nos da un plan para poder alcanzar nuestro goal, dadas nuestras condiciones iniciales:

```
ff: found legal plan as follows
step  0: COGER HAB0 P1 R1
      1: MOVER R1 HAB0 HAB1 FL2 FL1
      2: MOVER R1 HAB1 HAB2 FL1 FL0
      3: DEJAR HAB2 P1 R1
      4: COGER HAB2 P2 R1
      5: MOVER R1 HAB2 HAB1 FL0 FL2
      6: MOVER R1 HAB1 HAB0 FL2 FL1
      7: DEJAR HAB0 P2 R1
      8: MOVER R1 HAB0 HAB1 FL1 FL0
```

Figura 2.3: plan calculado para problema Ejercicio2

que vemos que resuelve nuestro problema. Probemos ahora con el otro problema algo más complejo, a ver si igualmente, encuentra un plan. Recordemos que añadíamos para ello tres elementos más: un robot, un paquete y una habitación, como se describe en el fichero *problemaEjercicio2mod.pddl* (incluido en el .zip) y vemos que igualmente nos da un plan solución(el mismo que nos daba en el ejercicio anterior pero mostrando los niveles de batería).

```
ff: found legal plan as follows
step  0: COGER HAB2 P2 R2
      1: MOVER R1 HAB0 HAB1 FL2 FL1
      2: MOVER R2 HAB2 HAB1 FL2 FL1
      3: MOVER R2 HAB1 HAB0 FL1 FL0
      4: DEJAR HAB0 P2 R2
      5: MOVER R2 HAB0 HAB1 FL0 FL2
      6: MOVER R1 HAB1 HAB0 FL1 FL0
      7: MOVER R2 HAB1 HAB0 FL2 FL1
      8: COGER HAB0 P1 R1
      9: COGER HAB0 P3 R2
     10: MOVER R2 HAB0 HAB1 FL1 FL0
     11: MOVER R1 HAB0 HAB1 FL0 FL2
     12: MOVER R1 HAB1 HAB2 FL2 FL1
     13: DEJAR HAB2 P1 R1
     14: MOVER R1 HAB2 HAB1 FL1 FL0
     15: MOVER R2 HAB1 HAB2 FL0 FL2
     16: MOVER R2 HAB2 HAB3 FL2 FL1
     17: DEJAR HAB3 P3 R2
     18: MOVER R2 HAB3 HAB2 FL1 FL0
     19: MOVER R2 HAB2 HAB1 FL0 FL2
```

Figura 2.4: plan calculado para problema Ejercicio2

3. Escribir un dominio de planificación en PDDL, modificando el dominio del anterior ejercicio, de manera que se puedan utilizar ahora dos acciones diferentes, moverse rápido y moverse lento tales que moverse rápido consume más unidades de fuel que moverse lento. Probarlo con varios problemas.

```
(define (domain RobotDistribuidor)
  (:requirements :typing)
  (:types objeto - object
    paquete robot - objeto
    habitacion flevel)

  (:predicates
    (at ?r - objeto ?h - habitacion)
    (conectada ?h1 - habitacion ?h2 - habitacion)
    (libre ?r - robot)
    (cogido ?paq - paquete ?r - robot)
    (batrestante ?r - robot ?bat - flevel)
    (cambio-lento ?n1 ?n2 - flevel)
    (cambio-rapido ?n1 ?n2 - flevel))

  (:action mover-lento
    :parameters (?r - robot ?origen ?destino - habitacion ?bat1 ?bat2 - flevel)
    :precondition (and
      (at ?r ?origen)
      (conectada ?origen ?destino)
      (batrestante ?r ?bat1)
      (cambio-lento ?bat1 ?bat2))
    :effect (and
      (at ?r ?destino)
      (not (at ?r ?origen))
      (not (batrestante ?r ?bat1))
      (batrestante ?r ?bat2)))

  (:action mover-rapido
    :parameters (?r - robot ?origen ?destino - habitacion ?bat1 ?bat2 - flevel)
    :precondition (and
      (at ?r ?origen)
      (conectada ?origen ?destino)
      (batrestante ?r ?bat1)
      (cambio-rapido ?bat1 ?bat2))
    :effect (and
      (at ?r ?destino)
      (not (at ?r ?origen))
      (not (batrestante ?r ?bat1))
      (batrestante ?r ?bat2)))

  (:action coger
    :parameters (?hab - habitacion ?paq - paquete ?r - robot)
    :precondition (and
      (at ?paq ?hab)
      (libre ?r)
      (at ?r ?hab))
    :effect (and
      (not (libre ?r))
      (cogido ?paq ?r)
      (not (at ?paq ?hab))))

  (:action dejar
    :parameters (?hab - habitacion ?paq - paquete ?r - robot)
    :precondition (and
      (cogido ?paq ?r)
      (at ?r ?hab))
    :effect (and
      (at ?paq ?hab)
      (libre ?r)
      (not (cogido ?paq ?r))))
```

Figura 3.1: Domininio solución para problema Ejercicio3

- Explicación: Seguimos trabajando sobre el mismo dominio, por lo que seguiremos teniendo robots y paquetes como objetos(heredan de object), y habitación y flevel.

Como **predicados** tenemos los mismos que hasta ahora, pero ahora tendremos dos predicados para cam-

bio:

1. **cambio-lento(n1,n2)** indica que el nivel de batería n1 cambia al nivel de batería n2 ,cuando esta se consume. Este cambio será lento (de unidad en unidad) , por tanto, este predicado es usado como precondition para que el robot se mueva lento.
2. **cambio-rapido(n1,n2)** indica que el nivel de batería n1 cambia al nivel de batería n2 ,cuando esta se consume. Este cambio será más brusco (de dos unidades en dos unidades), por tanto, este predicado es usado como precondition para que el robot se mueva rápido, pues consume más batería.

En cuanto a las **acciones** mantenemos las mismas que hasta ahora, pero añadiendo las de **mover- lento** y **mover-rapido**:

1. **mover-lento(robot,habitacionOrigen,habitacionDestino,bateriaActual,bateriaPosterior)** que tendrá como efecto, que el robot se desplaza de una habitación origen a una habitación de destino de forma lenta, siempre que el robot se encuentre en la habitación origen y que tenga batería. Como resultado de esta acción, el robot se desplazará a la siguiente habitación de forma más lenta pero también decrementará su nivel de batería menos.
2. **mover-rapido(robot,habitacionOrigen,habitacionDestino,bateriaActual,bateriaPosterior)** que tendrá como efecto, que el robot se desplaza de una habitación origen a una habitación de destino de forma rápida (siempre que el robot se encuentre en la habitación origen y que tenga batería). Como resultado de esta acción, el robot se desplazará a la siguiente habitación de forma más rápida pero también decrementará su nivel de batería más, pues hemos especificado con el predicado **cambio-rapido(...)** que moverse rápido implicará gastar el doble de batería que moverse lento.

Para el problema más simple que venimos tratando desde el principio:

```

(define (problem RDistribuye-1)
  (:domain RobotDistribuidor)
  (:objects
    r1 - robot
    p1 - paquete
    p2 - paquete
    hab0 - habitacion
    hab1 - habitacion
    hab2 - habitacion
    fl0 - flevel
    fl1 - flevel
    fl2 - flevel
  )
  (:init
    (libre r1)
    (batrestante r1 fl2) ; inicialmente tiene la batería cargada
    (at r1 hab0)
    (at p1 hab0)
    (at p2 hab2)
    (cambio-lento fl2 fl1) ; disminuye la batería
    (cambio-lento fl1 fl0) ; disminuye la batería
    (cambio-lento fl0 fl2) ; se carga la batería

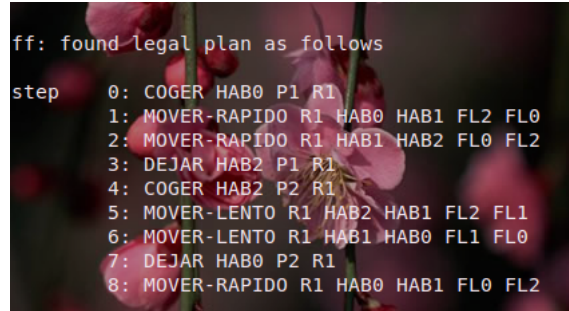
    (cambio-rapido fl2 fl0)
    (cambio-rapido fl0 fl2)

    (conectada hab0 hab1)
    (conectada hab1 hab0)
    (conectada hab2 hab1)
    (conectada hab1 hab2)
  )
  (:goal (and
    (at r1 hab1)
    (at p2 hab0)
    (at p1 hab2)
  ))
)

```

Figura 3.2: descripción problemaEjercicio3.pddl

y ejecutando, vemos que el planificador nos da el plan poder alcanzar nuestro goal:



```

ff: found legal plan as follows

step  0: COGER HAB0 P1 R1
       1: MOVER-RAPIDO R1 HAB0 HAB1 FL2 FL0
       2: MOVER-RAPIDO R1 HAB1 HAB2 FL0 FL2
       3: DEJAR HAB2 P1 R1
       4: COGER HAB2 P2 R1
       5: MOVER-LENTO R1 HAB2 HAB1 FL2 FL1
       6: MOVER-LENTO R1 HAB1 HAB0 FL1 FL0
       7: DEJAR HAB0 P2 R1
       8: MOVER-RAPIDO R1 HAB0 HAB1 FL0 FL2

```

Figura 3.3: plan calculado para problema Ejercicio3

Si probamos con el otro problema algo más complejo, vemos que también encuentra un plan. (Fichero *problemaEjercicio3mod.pddl* incluido en el .zip):

```
ff: found legal plan as follows
step  0: COGER HAB2 P2 R2
      1: MOVER-RAPIDO R2 HAB2 HAB1 FL2 FL0
      2: MOVER-RAPIDO R2 HAB1 HAB0 FL0 FL2
      3: COGER HAB0 P1 R1
      4: DEJAR HAB0 P2 R2
      5: COGER HAB0 P3 R2
      6: MOVER-RAPIDO R2 HAB0 HAB1 FL2 FL0
      7: MOVER-RAPIDO R1 HAB0 HAB1 FL2 FL0
      8: MOVER-RAPIDO R1 HAB1 HAB2 FL0 FL2
      9: DEJAR HAB2 P1 R1
     10: MOVER-LENTO R1 HAB2 HAB1 FL2 FL1
     11: MOVER-RAPIDO R2 HAB1 HAB2 FL0 FL2
     12: MOVER-RAPIDO R2 HAB2 HAB3 FL2 FL0
     13: DEJAR HAB3 P3 R2
     14: MOVER-RAPIDO R2 HAB3 HAB2 FL0 FL2
     15: MOVER-RAPIDO R2 HAB2 HAB1 FL2 FL0
```

Figura 3.4: plan calculado para problema Ejercicio3 con 2 robots,4 habitaciones,3 paquetes y 2 velocidades de movimiento

4. Escribir un dominio de planificación en PDDL 2.1 que responda a los requisitos explicados en los anteriores ejercicios, utilizando las capacidades expresivas de PDDL 2.1, es decir, representando funciones numéricas. Al igual que en los anteriores ejercicios, definir distintos problemas para comprobar que vuestro dominio es correcto.

```
(define (domain RobotDistribuidor)
  (:requirements :typing)
  (:types objeto - object
    paquete robot - object
    habitacion ;numero - integer
  )
  (:predicates
    (at ?r - objeto ?h - habitacion)
    (conectada ?h1 - habitacion ?h2 - habitacion)
    (libre ?r - robot)
    (cogido ?paq - paquete ?r - robot)
    ;(cambio-lento ?n1 ?n2 - numero)
    ;(cambio-rapido ?n1 ?n2 - numero)
  )

  (:functions
    (batrestante ?r - robot))

  (:action cargarbat
    :parameters (?r - robot)
    :precondition: (<=(batrestante ?r) 25)
    :effect (increase (batrestante ?r) 50))

  (:action mover-lento
    :parameters (?r - robot ?origen ?destino - habitacion)
    :precondition (and
      (at ?r ?origen)
      (conectada ?origen ?destino)
      (>=(batrestante ?r) 25))
    ;(cambio-lento ?bat1 ?bat2)
    :effect (and
      (at ?r ?destino)
      (not (at ?r ?origen))
      (decrease (batrestante ?r) 25)))

  (:action mover-rapido
    :parameters (?r - robot ?origen ?destino - habitacion)
    :precondition (and
      (at ?r ?origen)
      (conectada ?origen ?destino)
      (>=(batrestante ?r) 50))
    :effect (and
      (at ?r ?destino)
      (not (at ?r ?origen))
      (decrease (batrestante ?r) 50)))

  (:action coger
    :parameters (?hab - habitacion ?paq - paquete ?r - robot)
    :precondition (and
      (at ?paq ?hab)
      (libre ?r)
      (at ?r ?hab))
    :effect (and
      (not (libre ?r))
      (cogido ?paq ?r)
      (not (at ?paq ?hab))))

  (:action dejar
    :parameters (?hab - habitacion ?paq - paquete ?r - robot)
    :precondition (and
      (cogido ?paq ?r)
      (at ?r ?hab))
    :effect (and
      (at ?paq ?hab)
      (libre ?r)
      (not (cogido ?paq ?r))))
```

Figura 4.1: dominio solución propuesto al ejercicio4

- Explicación: Seguimos trabajando sobre el mismo dominio, por lo que seguiremos teniendo robots y paquetes como objetos (heredan de object), y habitaciones, sin embargo, como ahora vamos a definir funciones numéricas, no necesitaremos los flevel.

Como **predicados** tenemos los mismos que hasta ahora, pero ahora ya no necesitamos los predicados : cambio-lento y cambio-rápido, pues gestionaremos los cambios del nivel de la batería con funciones.

funciones:

1. **batrestante(robot)** nos creamos esta función **numérica** para llevar la gestión del nivel de batería restante del robot, en todo momento. Como se especifica en la descripción del problema, inicialmente ésta siempre se considerará que está plenamente cargada.

acciones mantenemos las mismas que hasta ahora, modificando **mover-lento** y **mover-rápido** y añadiendo una nueva acción de recarga de batería:

1. **mover-lento(robot,habitacionOrigen,habitacionDestino)** ya no recibe como parámetro la bateríaActual y la bateríaPosterior, pues como ahora tratamos con funciones, simplemente tendremos en cuenta como precondition que para moverse lento se necesita, al menos, un 25 % de batería , y que tras realizar un movimiento el robot se descargará un 25 %.
2. **mover-rápido(robot,habitacionOrigen,habitacionDestino)** Análogo al caso anterior, sólo que ésta acción consumirá más batería y por tanto tendrá como precondition que el robot tenga, al menos, un 50 % de batería y como efecto que tras haberse desplazado el robot se descargará un 50 %.
3. **cargarbat(robot)** esta acción es nueva, y se define como la acción de recargar la batería del robot. Será posible recargar siempre que la batería del robot sea igual o menor del 25 % (precondition) y tendrá como efecto que se incrementa en un 50 % el nivel de la batería del robot. Para que el robot pueda tener la batería cargada completamente, será necesario que recarge varias veces.

Y Vemos que para el problema básico que venimos considerando en todos los apartados (un robot, 3 habitaciones, 2 paquetes), de nombre *problemaEjercicio4.pddl*, obtenemos como resultado un plan con 9 pasos.

```

(define (problem RDistribuye-1)
  (:domain RobotDistribuidor)
  (:objects
    r1 - robot
    p1 - paquete
    p2 - paquete
    hab0 - habitacion
    hab1 - habitacion
    hab2 - habitacion
  )
  (:init
    (libre r1)
    (= (batrestante r1) 100) ; inicialmente tiene la batería cargada
    (at r1 hab0)
    (at p1 hab0)
    (at p2 hab2)
    ; (cambio-lento 100 75) ; disminuye la batería
    ; (cambio-lento 75 50) ; disminuye la batería
    ; (cambio-lento 50 25)
    ; (cambio-lento 25 0)
    ; (cambio-lento 0 100)
    ; (cambio-rapido fl2 fl0)
    ; (cambio-rapido fl0 fl2)

    (conectada hab0 hab1)
    (conectada hab1 hab0)
    (conectada hab2 hab1)
    (conectada hab1 hab2)
  )
  (:goal (and
    (at r1 hab1)
    (at p2 hab0)
    (at p1 hab2)
  ))
)

```

Figura 4.2: descripción problemaEjercicio4.pddl

```
ff: found legal plan as follows
step  0: COGER HAB0 P1 R1
      1: MOVER-RAPIDO R1 HAB0 HAB1
      2: MOVER-LENTO R1 HAB1 HAB2
      3: DEJAR HAB2 P1 R1
      4: COGER HAB2 P2 R1
      5: MOVER-LENTO R1 HAB2 HAB1
      6: CARGARBAT R1
      7: MOVER-LENTO R1 HAB1 HAB0
      8: DEJAR HAB0 P2 R1
      9: MOVER-LENTO R1 HAB0 HAB1
```

Figura 4.3: plan solución propuesto al problemaEjercicio4

Por último, probamos con el otro problema que venimos considerando a lo largo de esta memoria, en el que consideramos dos robots, cuatro habitaciones y tres paquetes (fichero *problemaEjercicio4mod.pddl*) obteniendo el plan de 17 pasos:

```
ff: found legal plan as follows
step  0: COGER HAB2 P2 R2
      1: MOVER-RAPIDO R2 HAB2 HAB1
      2: MOVER-LENTO R2 HAB1 HAB0
      3: COGER HAB0 P3 R1
      4: DEJAR HAB0 P2 R2
      5: COGER HAB0 P1 R2
      6: MOVER-RAPIDO R1 HAB0 HAB1
      7: MOVER-LENTO R2 HAB0 HAB1
      8: CARGARBAT R2
      9: MOVER-LENTO R2 HAB1 HAB2
     10: DEJAR HAB2 P1 R2
     11: MOVER-LENTO R2 HAB2 HAB1
     12: MOVER-LENTO R1 HAB1 HAB2
     13: MOVER-LENTO R1 HAB2 HAB3
     14: DEJAR HAB3 P3 R1
     15: CARGARBAT R1
     16: MOVER-LENTO R1 HAB3 HAB2
     17: MOVER-LENTO R1 HAB2 HAB1
```

Figura 4.4: plan solución propuesto al problemaEjercicio4mod.pddl