PLANIFICACIÓN DEL DOMINIO “ROBOTS”

Alumnos: Daniel Hernán Hidalgo Aliena, Raúl García Crespo

1. Descripción del dominio

El problema que presenta el dominio “Robots” viene definido por los siguientes enunciados:

- Se tiene un escenario representado por distintas habitaciones.
- Cierto número de personas y robots se encuentran repartidos por las habitaciones que componen el escenario.
- Los robots pueden desplazarse entre habitaciones adyacentes, pero solo cuando tengan permiso para entrar en la habitación de destino. Las personas permanecen en la misma habitación durante todo el tiempo.
- Los robots deben servir un té a cada una de las personas presentes en el problema, para ello deberán desplazarse por las habitaciones, recogerán tazas vacías de un armario situado en una habitación, prepararán el té en la máquina situada en la misma u otra habitación y servirán la taza preparada a una persona.
- Cada robot puede cargar con 2 tazas como máximo y puede dar o recibir tazas de otro robot siempre que se mantenga este límite.
- El problema se considera resuelto cuando a todas las personas presentes se les ha servido una taza de té.

La forma base del problema con la que se ha trabajado durante el desarrollo de esta práctica se compone de 2 robots, 2 personas y 5 habitaciones siguiendo la distribución inicial que pude verse en la figura:

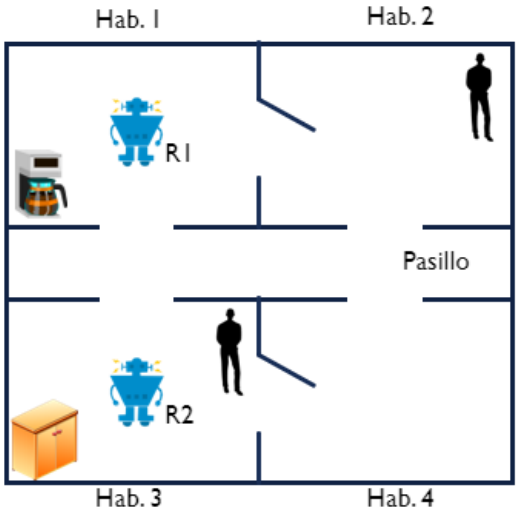


Figura 1: Distribución inicial del problema base

2. Dominio proposicional

2.1. Evolución de nuestra solución

Durante el desarrollo de los ficheros pddl del dominio y el problema, los predicados y acciones que empleamos fueron evolucionando a medida que avanzábamos y veíamos que funcionaba y que no.

En un principio comenzamos desarrollando el dominio sin hacer distinción de con que brazo un robot sujetaba una taza o si tenía una mano libre para coger otra taza con idea de que así sería más sencillo generalizar las acciones, simplemente empleando el predicado:

```
(sostiene ?x - robot ?y - taza)
```

Sin embargo este planteamiento presentaba problemas a la hora de definir las precondiciones de las acciones que involucraban el manejo de tazas, por lo que acabamos optando por incluir predicados que indicasen que sostenía cada robot en cada mano:

```
(sostieneIzda ?x - robot ?y - taza)
(sostieneDcha ?x - robot ?y - taza)
(libreIzda ?x -robot)
(libreDcha ?x -robot)
```

Dado que en un principio pensamos en evitar los predicados con más de 2 argumentamos, desarrollamos esto 4 predicados para representar las manos de los robots y su contenido. Esta representación generaba nuevos problemas puesto que ahora eran necesarias 4 acciones para intercambiar una taza entre 2 robots, una por cada par de manos (Der Izq, Der Der, Izq Izq e Izq Der). Lo mismo sucedía para la acción de servir té.

Para resolver esta situación en la que existían tantas acciones innecesarias añadimos nuevos tipos a nuestro problema: el tipo “mano”, que podía tomar los valores IZQ o DER; y el tipo “vacio”, que solo podía tomar el valor VACIO. Dado que los valores de estos tipos siempre serían los mismos en cualquier posible instanciación del problema, los definimos como constantes en el dominio. Adicionalmente y para reducir el número de predicados necesarios decidimos emplear un solo predicado de 3 argumentos que representase que sostenía un robot en una mano, independientemente de si era una taza o tenía la mano libre:

```
(sostiene ?x - robot ?y - (either taza vacio) ?z - mano)
```

Con esto logramos reducir y simplificar el número de acciones hasta su versión definitiva. En los siguientes apartados se explican en detalle los tipo, predicados y acciones que componen la versión final de nuestro dominio proposicional y el cual se ha empleado en el desarrollo de los dominios temporales y de recursos.

2.2. Tipos

A continuación se listan los tipos que hemos empleado y una descripción de su propósito:

robot	Representa a cada uno de los robots del problema. Se instancian tantos valores de este tipo como robots haya en el problema.
person	Representa a cada una de las personas del problema. Se instancian tantos valores de este tipo como personas haya en el problema.
room	Representa cada una de las habitaciones que componen el escenario. Se instancian tantos valores de este tipo como habitaciones haya en el problema.
taza	Representa cada una de las tazas del problema. Ya que nuestra versión del dominio requiere que las tazas estén explícitamente declaradas, es necesario instanciar tantos valores de este tipo como tazas vayan a ser utilizadas, es decir, una taza por cada persona presente en el problema.
mano	Representa cada mano de cada robot. Solo puede poseer los valores DER o IZQ los cuales se definen como constantes del dominio.
vacio	Representa el contenido de una mano cuando esta no sostiene una taza. Solo puede tomar el valor VACIO el cual se define como constante del dominio.

En las siguientes imágenes se incluyen los fragmentos de código de la definición de estos tipos en el dominio y su instanciación para el problema base:

```
(define (domain robots)
  (:requirements :typing)
  (:types robot person room taza mano vacio)
  (:constants
        VACIO - vacio
        IZQ DER - mano
  )
  (:predicates
```

Figura 2: Definición de tipos

```
(define (problem probrobots)
(:domain robots)
(:objects T1 T2 - taza
          P1 P2 - person
          R1 R2 - robot
          H1 H2 H3 H4 PASILLO - room)
```

Figura 3: Instanciación de tipos

2.3. Predicados

A continuación se litan los predicados que hemos empleado junto a una descripción de sus funciones:

- **(adjacente ?x - room ?y - room)**:
  Este predicado representa la relación de adyacencia entre dos habitaciones, es decir, representa la posibilidad de desplazarse desde una habitación a otra al estar conectadas. Dado que pddl hace distinción en el orden de los argumentos, necesitamos instanciar este predicado 2 veces por cada par de habitaciones conectadas, una para los desplazamientos de la habitación ?x a la habitación ?y, y otra para los desplazamientos de ?y a ?x.
- **(servido ?x - person):**
  Este predicado se genera cuando la persona ?x ha recibido su taza té y se emplea en la definición del “goal” del problema.
- **(estaEn ?x - (either robot person) ?y - room):**
  Este predicado representa tanto la posición de los robots como de las personas en una habitación del problema. Las personas mantienen el mismo predicado asociado a su posición durante todo el problema, mientras que el de los robots se modifica en la acción correspondiente a su desplazamiento.
- **(permitido ?x - robot ?y - room):**
  Un robot solo puede acceder de una habitación a otra en caso de que ambas habitaciones estén conectadas y dicho robot posea permiso para entrar en la habitación de destino. Este predicado representa las habitaciones en las que el robot ?x posee acceso. Se instancia tantas veces como habitaciones en las que el robot puede desplazarse y por cada robot del problema.
- **(sostiene ?x - robot ?y - (either taza vacio) ?z - mano):**
  Este predicado representa el contenido de una mano de uno de los robots. Como ya se ha comentado en el apartado anterior, un robot puede sostener una taza o tener la mano libre (representado por el tipo “vacio”). Se instanciará inicialmente 2 veces por cada robot, una para cada posible valor del tipo mano, con el parámetro ?y igual a VACIO e irá evolucionando a medida que los robots cojan e intercambien tazas.

- **(tazaLlena ?x - taza) y (tazaVacia ?x - taza):**
  Ya que es necesario explicitar las tazas, cada una de ellas llevará asociado en todo momento uno de estos dos predicados, representando si están vacías o contienen té. Inicialmente siempre tendrán asociado el predicado **(tazaVacia ?x - taza).**
- **(maquinaTe ?x - room) y (armario ?x - room):**
  Representan la posición en una habitación del armario que contiene todas las tazas inicialmente y la máquina necesaria para preparar el té.
- **(armarioTiene ?x - taza):**
  Este último predicado representa el contenido del armario cuya posición se ha definido con el predicado **(armario ?x - room).** Inicialmente existirán tantos de estos predicados como tazas existan en el problema e irán desapareciendo a medida que los robots cojan las tazas del armario.

2.4. Acciones

A continuación se presentan las acciones definidas para este dominio. Recordemos que según la definición del problema los robots deben ser capaces de desplazarse, coger tazas del armario, preparar té, dar o recibir tazas de otros robots y ser capaces de servir una taza de té a las personas.

2.4.1. Acción: desplazarRobot

La acción más básica y necesaria es el desplazamiento de un robot de una habitación a otra, ya que de otro modo solo podría resolverse el problema en caso de que el robot, la persona, el armario y la máquina de té estuviesen todos en la misma habitación. Nuestra acción de desplazamiento recibe 3 parámetros: 1 robot y 2 habitaciones, una de partida y otra de destino.

Como precondiciones de esta acción son necesarios los enunciados que declaren que el robot recibido como parámetro se encuentra en la habitación de partida **(estaEn ?x - (either robot person) ?y - room),** que dicho robot posee permiso para acceder a la habitación de destino **(permitido ?x - robot ?y - room),** y que ambas habitaciones se encuentran conectadas **(adjacente ?x - room ?y - room).** Si se cumplen estas condiciones, se ejecuta la acción y el predicado **(estaEn ?x - (either robot person) ?y - room)** se sustituye para que represente al robot en la habitación de destino.

```
(:action desplazarRobot
  :parameters (?robot1 - robot ?room1 - room ?room2 - room)
  :precondition (and (estaEn ?robot1 ?room1)(adjacente ?room1 ?room2)(permitido ?robot1 ?room2))
  :effect (and (not (estaEn ?robot1 ?room1)) (estaEn ?robot1 ?room2))
)
```

Figura 4: Código de la acción "desplazarRobot"

2.4.2. Acción: cogerTaza

Esta acción simboliza el proceso por el que un robot coge una taza desde el armario que las contiene inicialmente. Para ello recibe 4 parámetros: una habitación, un robot, una taza y una mano. Las precondiciones para ejecutar esta acción son las siguientes:

- Tanto el robot como el armario deben encontrarse en la misma habitación. Es decir se deben satisfacer los predicados **(estaEn ?x - (either robot person) ?y - room)** para el robot adecuado y **(armario ?x - room)** con el mismo valor del argumento de tipo “room”.
- El armario debe contener la taza, indicado con el predicado **(armarioTiene ?x - taza).**
- Al menos una de las manos del robot debe estar libre. Para ello la acción deberá encontrar un predicado del tipo **(sostiene ?x - robot ?y - (either taza vacio) ?z - mano)** donde el parámetro ?y sea igual a VACIO para el robot.

Una vez se satisfagan están condiciones la acción se ejecutará eliminando el predicado que representa la posición de la taza en el armario y sustituirá el predicado por el que el robot tiene la mano libre por uno en el que sostiene la taza que ha cogido del armario.

```
(:action cogerTaza
  :parameters (?robot1 - robot ?room1 - room ?taza1 - taza ?mano1 - mano)
  :precondition (
      and
        (estaEn ?robot1 ?room1)
        (armario ?room1)
        (armarioTiene ?taza1)
        (sostiene ?robot1 VACIO ?mano1))
  :effect (
      and
        (not (armarioTiene ?taza1))
        (not (sostiene ?robot1 VACIO ?mano1))
        (sostiene ?robot1 ?taza1 ?mano1))
)
```

Figura 5: Código de la acción "cogerTaza"

2.4.3. Acción: darTaza

Esta acción representa el intercambio de tazas entre robots. Recibe como parámetros 2 robots, 1 habitación, 1 taza y 2 manos; y debe de cumplir las siguientes precondiciones para realizar su ejecución:

- Ambos robots deben encontrarse en la misma habitación, por lo tanto deben existir dos predicados del tipo **(estaEn ?x - (either robot person) ?y - room),** uno para cada robot y que coincidan en el valor de ?y.
- Uno de los robots debe sostener la taza que se va a intercambiar y el otro debe tener al menos una mano vacía, por lo tanto deben existir dos predicados del tipo **(sostiene ?x - robot ?y - (either taza vacio) ?z - mano);** uno asociado al primer robot con el parámetro ?y igual a la taza con la que se realiza el intercambio y otro asociado al otro robot con ?y igual a VACIO. El valor de los parámetros "mano" es indiferente.

Al final de la ejecución de esta acción los predicados del tipo **(sostiene ?x - robot ?y - (either taza vacio) ?z - mano)** se habrán sustituido por otros predicados del mismo tipo en los que el robot que antes sostenia la taza ahora tiene una mano libre y el robot que tenía la mano libre ahora sostiene una taza.

```
(:action darTaza
  :parameters (?robot1 - robot ?robot2 - robot ?room1 - room ?taza1 - taza ?mano1 - mano ?mano2 - mano)
  :precondition (
      and
        (estaEn ?robot1 ?room1)
        (estaEn ?robot2 ?room1)
        (sostiene ?robot1 ?taza1 ?mano1)
        (sostiene ?robot2 VACIO ?mano2))
  :effect (
      and
        (not (sostiene ?robot1 ?taza1 ?mano1))
        (sostiene ?robot1 VACIO ?mano1)
        (not (sostiene ?robot2 VACIO ?mano2))
        (sostiene ?robot2 ?taza1 ?mano2))
)
```

Figura 6:Código de la acción "darTaza"

2.4.4. Acción: hacerTe

Esta acción representa el proceso de preparar una taza de té en la máquina por un robot. Para su ejecución recibe 4 parámetros: 1 robot, 1 habitación, 1 taza y 1 mano; y deben cumplirse las siguientes precondiciones:

- Al igual que sucedía en la acción "cogerTaza" con el robot y el armario de tazas, en este caso el robot y la máquina de té deben encontrarse en la misma habitación. Para ello deben existir dos predicados, uno para la posición del robot y otro para la de la máquina en los el valor de la habitación sea el mismo.
- El robot debe de sostener la taza en una de sus manos, por lo tanto debe existir un predicado **(sostiene ?x - robot ?y - (either taza vacio) ?z - mano)** para este robot con ?y igual a una taza.

- La taza que sostiene el robot debe estar necesariamente vacía para poder preparar el té en ella, por lo que debe existir una predicado del tipo **(tazaVacia ?x - taza)** asociado a dicha taza.

Una vez ejecutada esta acción el único cambio en los predicados será el sustituir el predicado **(tazaVacia ?x - taza)** asociado a la taza usada en la acción por el predicado **(tazaLlena ?x - taza)**.

```
(:action hacerTe
  :parameters (?robot1 - robot ?room1 - room ?taza1 - taza ?mano1 - mano)
  :precondition (
      and
        (estaEn ?robot1 ?room1)
        (sostiene ?robot1 ?taza1 ?mano1)
        (tazaVacia ?taza1)
        (maquinaTe ?room1))
  :effect (
      and
        (not(tazaVacia ?taza1))
        (tazaLlena ?taza1))
)
```

Figura 7: Código de la acción "hacerTe"

2.4.5. Acción: servirTe

Por último, tenemos la acción servirTe, la cual simula el proceso por el que un robot da una taza de té ya preparada a una persona. Antes de ejecutarse deben cumplirse las precondiciones:

- Tanto el robot como la persona deben encontrarse en la misma habitación, para ello se deben buscar 2 predicados de posición asociados al robot y a la persona con el mismo valor del parámetro que representa la habitación.
- El robot debe sostener una taza llena, por lo que serán necesarios 2 predicados: un predicado **(sostiene ?x - robot ?y - (either taza vacio) ?z - mano)** en el que se especifica que el robot sostiene una taza en una de sus manos; y un predicado **(tazaLlena ?x - taza)** asociado a la taza que sostiene el robot.
- Por la forma en la que se ha definido el “goal” de este dominio, no es necesario comprobar en las precondiciones que la persona ya haya sido servido, pues el propio planificador ya deduce que servir dos veces a la misma persona no es adecuado para la resolución del problema.

Una vez ejecutada la acción, el predicado **(sostiene ?x - robot ?y - (either taza vacio) ?z - mano)** empleado se sustituirá por uno en el que el robot tenga la mano libre y se incluirá el predicado **(servido ?x - person)** para indicar que esa persona ya ha sido servida.

2.5. Definición del “goal”

Como ya hemos dicho, el problema del dominio “robots” se considera resuelto cuando todas las personas presentes en el problema han sido servidas. Por lo tanto el “goal” será tan simple como obtener un estado en el que cada persona tenga asociado un predicado **(servido ?x - person).** Para el caso del problema base en el que solo existen 2 personas:

```
(:goal (and (servido P1)(servido P2)))
```

3. Dominio temporal

El propósito de este apartado de la práctica es el de añadir modificaciones al dominio proposicional para que incluya acciones durativas y el objetivo del planificador sea hallar el plan que menor tiempo consuma. Para ello el nuevo dominio debe cumplir lo siguiente:

- La duración de los desplazamientos de robots entre habitaciones dependerá de la distancia entre habitaciones y la velocidad del robot, esta última debe ser distinta entre los robots.
- La duración de las acciones de dar tazas dependerán del peso de la taza, siendo la duración mayor cuando la taza esté llena.
- Las acciones de coger una taza del armario y preparar té en la máquina tendrán ambas una duración fija de 2 unidades de tiempo.

Para adaptar el problema a estas condiciones y partiendo del dominio proposicional desarrollado en el apartado anterior los cambios que han sido necesarios se listan en los siguientes aparatados:

3.1. Funciones

Para definir la duración de las acciones primero es necesario definir un conjunto de funciones que nos devuelvan los valores numéricos correspondientes por ejemplo a la distancia entre dos habitaciones.

- **(distancia ?r1 - room ?r2 - room)**: Devuelve la distancia entre dos habitaciones. Al igual que ocurre con el predicado **(adjacente ?x - room ?y - room)** es necesario instanciar esta función por duplicado para cada par de habitaciones conectadas, una la primera habitación a la segunda y otra a la inversa.
- **(velocidad ?r1 - robot)**: Devuelve el valor de la velocidad de uno de los robots. Es necesario instanciar esta función para cada robot.
- **(darTazaLlena) y (darTazaVacia)**: Devuelven la duración de la acción “darTaza” dependiendo de si la taza que se da esta vacía o llena respectivamente. Ambas funciones se instancian solo una vez por problema.

3.2. Cambios en las acciones

A parte de los cambios gramaticales que conlleva incluir acciones durativas como sustituir “precondition” por “condition” en nuestras acciones, se han realizado diferentes cambios a las acciones del dominio proposicional:

3.2.1. Acción: “desplazarRobot”

En esta acción se ha incluido la cláusula “duration” de manera que esta dependa de los resultados de las funciones **(distancia ?r1 - room ?r2 - room)** y **(velocidad ?r1 - robot)** para los parámetros escogidos por la acción. La duración se ha representado en forma de cociente de manera que aumentará cuanto mayor sea la distancia entre habitaciones y disminuirá cuanto mayor sea la velocidad del robot.

```
(:durative-action desplazarRobot
  :parameters (?robot1 - robot ?room1 - room ?room2 - room)
  :duration (= ?duration (/ (distancia ?room1 ?room2) (velocidad ?robot1)))
  :condition (
      and (at start(estaEn ?robot1 ?room1))
      (at start(adjacente ?room1 ?room2))
      (at start(permitido ?robot1 ?room2))
  )
  :effect (
      and (at start(not (estaEn ?robot1 ?room1)))
      (at end(estaEn ?robot1 ?room2))
  )
)
```

Figura 8: Código de la acción durativa "desplazarRobot"

Adicionalmente, en las condiciones se han definido con la etiqueta “at Start” todas las precondiciones que deben cumplirse, ya que el robot solo debe estar en la habitación de partida al inicio de la acción y el resto de predicados en las precondiciones no pueden variar en nuestro dominio.

En los efectos sin embargo se ha etiquetado el predicado **(estaEn ?robot1 ?room1)** como “at Start” dado que en el momento en el que el robot inicia su movimiento este deja de estar en la habitación de partida; pero solo alcanza la habitación de destino al final de la duración de la acción, etiquetada con “at end”.

3.2.2. Acción: “cogerTaza”

En este caso se ha modificado la acción para que su duración sea siempre fija en 2 unidades de tiempo. Todas las precondiciones se han etiquetado “at start” ya que solo son ciertas al comienzo de la ejecución dela acción o no varían, con la excepción del predicado **(estaEn ?robot1 ?room1)** el cual se ha etiquetado con “overa all” ya que el robot debe permanecer en la misma habitación durante toda la duración de la acción.

En los efectos hemos definido que la taza deja de estar en el armario al iniciar la acción, que el robot deja de tener la mano libre también al comienzo de la acción y que el robot solamente sostiene la taza al finalizar la acción.

```
(:durative-action cogerTaza
  :parameters (?robot1 - robot ?room1 - room ?taza1 - taza ?mano1 - mano)
  :duration (= ?duration 2)
  :condition (
      and (over all(estaEn ?robot1 ?room1))
      (at start(armario ?room1))
      (at start(armarioTiene ?taza1))
      (at start(sostiene ?robot1 VACIO ?mano1))
  )
  :effect (
      and (at start(not (armarioTiene ?taza1)))
      (at start(not (sostiene ?robot1 VACIO ?mano1)))
      (at end(sostiene ?robot1 ?taza1 ?mano1))
  )
)
```

Figura 9: Código de la acción durativa "cogerTaza"

3.2.3. Acciones: darTazaV y darTazaLl

Para que nuestro dominio sea capaz de asignar duraciones distintas a la acción de dar una taza es necesario separar esta acción del dominio proposicional en dos acciones en el dominio temporal. Una de estas dos acciones incluirá en sus condiciones el predicado asociado a la taza indicando que esta se encuentra vacía, mientras que la otra acción incluirá la misma condición pero para la taza llena. La duración de ambas acciones será igual al valor de las funciones **(darTazaLlena)** y **(darTazaVacia)** según corresponda.

Para asegurar que ambos robots permanecen en la misma habitación durante toda la duración de la acción se han etiquetado los predicados **(estaEn ?robot1 ?room1)** y **(estaEn ?robot2 ?room1)** como “over all”, al igual que al predicado **(tazaVacia ?taza1)** para garantizar que, por ejemplo, no se ejecute la acción de preparar té sobre esa taza si los robots se encuentran en una habitación con máquina de té. El resto de precondiciones reciben la etiqueta “at start”.

En las condiciones finales, los predicados que desaparecen lo hacen al inicio de la acción, mientras que los que se añaden se incluyen al finalizarla.

```
(:durative-action darTazaLl
  :parameters (?robot1 - robot ?robot2 - robot ?room1 - room ?taza1 - taza ?mano1 - mano ?mano2 - mano)
  :duration (= ?duration (darTazaLlena))
  :condition (
     and (over all(estaEn ?robot1 ?room1))
     (over all(estaEn ?robot2 ?room1))
     (at start(sostiene ?robot1 ?taza1 ?mano1))
     (at start(sostiene ?robot2 VACIO ?mano2))
     (over all(tazaLlena ?taza1))
  )
  :effect (
     and (at start(not (sostiene ?robot1 ?taza1 ?mano1)))
     (at end(sostiene ?robot1 VACIO ?mano1))
     (at start(not (sostiene ?robot2 VACIO ?mano2)))
     (at end(sostiene ?robot2 ?taza1 ?mano2))
  )
)
```

Figura 10: Código de la acción durativa "darTazaLl"

3.2.4. Acción: hacerTe

La duración de esta acción se ha fijado en 2 unidades de tiempo y los únicos cambios que se han realizado sobre sus precondiciones es el etiquetado de cada precondición como “at start”, con la excepción del predicado correspondiente a la posición del robot ya que de nuevo esta debe mantenerse constante durante toda la acción con la etiqueta “over all”.

Las condiciones se han redefinido para que la taza deje de estar vacía al comienzo de la acción pero solo se la considera llena al finalizarla.

```
(:durative-action hacerTe
  :parameters (?robot1 - robot ?room1 - room ?taza1 - taza ?mano1 - mano)
  :duration (= ?duration 2)
  :condition (
      and(over all(estaEn ?robot1 ?room1))
      (at start(sostiene ?robot1 ?taza1 ?mano1))
      (at start(tazaVacia ?taza1))
      (at start(maquinaTe ?room1))
  )
  :effect (
      and (at start(not(tazaVacia ?taza1)))
      (at end(tazaLlena ?taza1))
  )
)
```

Figura 11: Código de la acción durativa "hacerTe"

3.2.5. Acción: servirTe

La duración de esta acción es igual a la duración de dar una taza llena, ya que nunca se podrá servir una taza vacía a una persona. En las condiciones de nuevo debemos controlar que el robot permanezca en la misma habitación durante toda la duración, y en los efectos se considera que la taza deja de estar sostenida por el robot al iniciar la acción, pero la persona estará servida y el robot tendrá la mano libre solo al final de la acción.

```
(:durative-action servirTe
  :parameters (?robot1 - robot ?room1 - room ?person1 - person ?taza1 - taza ?mano1 - mano)
  :duration (= ?duration (darTazaLlena))
  :condition (
      and (over all(estaEn ?robot1 ?room1))
      (over all(estaEn ?person1 ?room1))
      (at start(sostiene ?robot1 ?taza1 ?mano1))
      (at start(tazaLlena ?taza1))
  )
  :effect (
      and (at start(not (sostiene ?robot1 ?taza1 ?mano1)) )
      (at end(sostiene ?robot1 VACIO ?mano1))
      (at end(servido ?person1)))
)
```

Figura 12: Código de la acción durativa "servirTe"

3.3. Nueva métrica

Para que el planificador busque el plan que consuma menos tiempo no basta con definir la duración de las acciones, en el fichero pddl del problema, junto al “goal” es necesario definir una métrica para que se busque la minimización del tiempo de ejecución:

```
(:goal (and (servido P1)(servido P2)))

(:metric minimize(total-time))
```

4. Dominio con recursos numéricos

El propósito de este apartado de la práctica es el de añadir modificaciones al dominio temporal para que incluya el consumo de un recurso y poder comparar los planes generados con un recurso ilimitado y uno renovable. Hemos escogido como recurso una batería que permite que los robots se desplacen más deprisa.

En el caso del recurso no renovable, cada robot tendrá acceso a una batería ilimitada. En cambio, en el caso del recurso renovable, cada robot dispondrá de su propia batería que podrá agotarse. Esta se podrá recargar en una de varias estaciones de recarga, las cuales deberán situarse en una de las habitaciones.

Para ello el nuevo dominio debe cumplir lo siguiente:

- La duración de los desplazamientos de robots entre habitaciones dependerá además del consumo de batería del robot. Esta duración se reducirá a la mitad de un desplazamiento normal si consume batería.
- En el caso renovable. deberá haber al menos una estación de recarga accesible para cada robot. Solo podrá haber una estación por habitación, y varios robots podrán utilizar la misma estación simultáneamente.
- En el caso renovable, un robot no podrá realizar acciones en la misma habitación mientras esté recargando su batería.

Para adaptar el problema a estas condiciones y partiendo del dominio temporal desarrollado en el apartado anterior, los cambios que han sido necesarios se listan en los siguientes aparatados:

4.1. Predicados

Para representar la posibilidad de recargar la batería de un robot en estaciones de recarga, a diferencia del dominio temporal visto previamente, es necesario definir un nuevo conjunto de predicados.

- **(estacion ?r1 - room)**: Indica que existe una estación de recarga en la habitación indicada. Se debe instanciar predicados suficientes para que todos los robots tengan acceso al menos a una estación en su perímetro.
- **(disponible ?r1 - robot)**: Indica que el robot está disponible, es decir que no está recargando, para poder realizar acciones. Es necesario instanciar este predicado para cada robot al inicio del problema.
- **(recargando ?r1 - robot)**: Indica que el robot está recargando y no puede realizar ninguna otra acción. Es complementario a, y mutuamente exclusivo con el predicado **(disponible ?r1 - robot)**.

4.2. Funciones

Para representar el almacenamiento y el consumo de un recurso es necesario definir un nuevo conjunto de funciones que devuelvan el valor numérico correspondiente a la tasa de consumo y al gasto total.

- **(total-bateria-consumida)**: Devuelve el consumo total de batería de todos los robots del problema. Es necesario instanciar esta función con un valor de 0.
- **(bateria-maxima)**: Devuelve la capacidad máxima de batería que puede almacenar cada robot. Es necesario instanciar esta función una vez por problema.
- **(bateria ?r1 - robot)**: Devuelve la batería restante para el robot ?r1. Este valor se hallará entre 0 y **(bateria-maxima)**. Es necesario instanciar esta función una vez para cada robot en el rango previamente indicado.
- **(batería-por-segundo)**: Devuelve la cantidad de batería consumida por segundo al desempeñar un desplazamiento rápido. Es necesario instanciar esta función con un valor estrictamente positivo.
- **(recarga-por-segundo)**: Devuelve la cantidad de batería recargada por segundo al utilizar una estación de recarga. Es necesario instanciar esta función con un valor estrictamente positivo.

4.3. Cambios en las acciones

Para cumplir con los requisitos previamente establecidos se han realizado diferentes cambios a las acciones del dominio proposicional:

4.3.1. Acción: “recargar” (caso renovable)

Esta acción representa el proceso de acoplarse a una estación de recarga para recargar la batería de un robot hasta alcanzar la capacidad máxima. Recibe como parámetros 1 robot y 1 habitación; y debe de cumplirse que el robot se encuentre en una habitación que contenga una estación de recarga, y que no esté recargando ya.

En el caso de un recurso no renovable, no es necesario definir esta acción.

Para que se cumplan estas precondiciones se deben satisfacer durante toda la duración los predicados **(estaEn ?x - (either robot person) ?y - room)** para el robot adecuado y **(estacion ?x - room)** con el mismo valor del argumento de tipo “room”. Además, el predicado **(disponible ?x - robot)** para ese mismo robot deberá existir al inicio de la acción.

La duración de esta acción se calcula a partir de la diferencia entre las funciones **(bateria ?x - robot)** y **(bateria-maxima)**, es decir, la batería aún por recargar, y la tasa de recarga dada por **(recarga-por-segundo)**.

```
(:durative-action recargar
  :parameters (?robot1 - robot ?room1 - room)
  :duration (= ?duration (/ (- (bateria-maxima) (bateria ?robot1)) (recarga-por-segundo)))
  :condition (
    and (over all(estaEn ?robot1 ?room1))
    (over all(estacion ?room1))
    (at start (disponible ?robot1))
  )
  :effect (
    and (at start(not(disponible ?robot1)))
    (at start(recargando ?robot1))
    (at end(assign (bateria ?robot1) (bateria-maxima)))
    (at end(not (recargando ?robot1)))
    (at end(disponible ?robot1))
  )
)
```

Figura 14: Código de la acción durativa "recargar" en el caso renovable

Al principio de esta acción el robot pasará de estar disponible a estar recargando. Es decir, se eliminará el predicado **(disponible ?x - robot)** y se instanciará el predicado complementario **(recargando ?x - robot)**.

Al finalizar la duración de esta acción, la batería del robot se habrá recargado por completo y mediante la cláusula “assign” se le asignará a la función **(bateria ?x - robot)** su valor máximo, **(bateria-maxima)**. No se considera el estado de la batería en instantes intermedios ya que una vez un robot inicia una recarga, no puede ser interrumpida hasta su finalización.

Por último, el robot dejará de estar recargando y estará disponible de nuevo. Es decir, se eliminará el predicado **(recargando ?x - robot)** y se instanciará el predicado complementario **(disponible ?x - robot)** una vez más.

4.3.2. Precondición en todas las acciones (caso renovable)

En el caso de un recurso renovable, para que la acción **(:durative-action recargar…)** sea mutuamente exclusiva con cualquier otra acción, se debe añadir una precondición adicional a cada una de las demás acciones.

En el caso de un recurso no renovable, no es necesario añadir esta precondición.

Se debe comprobar para realizar una acción que el robot se encuentre disponible a lo largo de la duración mediante el predicado **(disponible ?x - robot)**. Para ello se etiqueta este predicado como “over all”, y se añade a la lista de precondiciones.

El resto de precondiciones no necesita ninguna modificación adicional.

4.3.3. Acción: “desplazarRobotTurbo”

Para que nuestro dominio sea capaz de asignar duraciones distintas al desplazamiento según el consumo de batería, es necesario separar la acción **(:durative-action desplazarRobot…)** del dominio temporal en dos acciones en el dominio con recursos numéricos. Es así como definimos una nueva acción **(:durative-action desplazarRobotTurbo…)** que representa un desplazamiento haciendo uso de la batería del robot.

En este caso, la duración del desplazamiento se calculará usando la misma fórmula que “desplazarRobot”, pero dividiéndola por 2 para cumplir con la reducción del tiempo necesario a la mitad.

```
(:durative-action desplazarRobotTurbo
  :parameters (?robot1 - robot ?room1 - room ?room2 - room)
  :duration (= ?duration (/ (/ (distancia ?room1 ?room2) (velocidad ?robot1)) 2))
  :condition (
    and (over all (disponible ?robot1))
    (at start(>= (bateria ?robot1) (* (/ (/ (distancia ?room1 ?room2) (velocidad ?robot1)) 2) (bateria-por-segundo))))
    (at start(estaEn ?robot1 ?room1))
    (at start(adjacente ?room1 ?room2))
    (at start(permitido ?robot1 ?room2))
  )
  :effect (
    and (at start(not (estaEn ?robot1 ?room1)))
    (at end(estaEn ?robot1 ?room2))
      (at end(decrease (bateria ?robot1) (* (bateria-por-segundo) (* (/ (/ (distancia ?room1 ?room2) (velocidad ?robot1)) 2) (bateria-por-segundo)))))
      (at end(increase (total-bateria-consumida) (* (bateria-por-segundo) (* (/ (/ (distancia ?room1 ?room2) (velocidad ?robot1)) 2) (bateria-por-segundo)))))
  )
)
```

Figura 13: Código de la acción durativa "desplazarRobotTurbo" en el caso renovable

Adicionalmente, para el caso de un recurso limitado y renovable se debe comprobar que el robot dispone de suficiente batería para terminar el desplazamiento sin que se agote su reserva, dada por **(bateria ?r1 - robot)**. En el caso de un recurso no renovable no se requiere esta nueva comprobación.

En los efectos hemos calculado el consumo de batería multiplicando la duración ya calculada y la función **(bateria-por-segundo)**. En el caso de un recurso no renovable, al finalizar la acción incrementamos la función **(total-bateria-consumida)** con el consumo calculado. En el caso de un recurso renovable, además de incrementar este contador total, se debe decrementar la propia batería del robot dada por la función **(bateria ?r1 - robot)**.

4.4. Nueva métrica

Para que el planificador busque el plan que consuma menos recursos ahora es necesario cambiar la función empleada para calcular la métrica a minimizar, haciendo uso de la función **(total-bateria-consumida)** que se ha actualizado a lo largo del plan:

```
(:goal (and (servido P1)(servido P2)))

(:metric minimize(total-bateria-consumida))
```

Es posible definir una métrica híbrida para buscar un equilibrio entre el tiempo de ejecución del plan y la cantidad de recurso consumido.

Una combinación lineal no es suficiente para equilibrar las dos métricas, ya que siempre será más eficiente minimizar una de las dos a costa de la otra.

Por eso escogemos una métrica híbrida basada en una suma de cuadrados:

$4(\text{total-time})^2 + (\text{total-bateria-consumida})^2$

```
(:goal (and (servido P1)(servido P2)))

(:metric minimize(+ (* 4 (* (total-time) (total-time) ) ) (* (total-bateria-consumida)
(total-bateria-consumida))))
```

Los coeficientes 4 y 1 (2^2 y 1^2) se han obtenido a partir del punto de equilibrio de la combinación lineal: *2(total-time) + (total-bateria-consumida)*.

5. Experimentos y Resultados

El propósito de este apartado es mostrar las variantes del problema que se han diseñado, así como los resultados obtenidos con los distintos planificadores para los dominios descritos previamente.

Base: Dominio proposicional básico
Temporal: Dominio temporal con **:durative-action**
Recursos: Dominios con recursos no renovable y renovable. 3 métricas:
- Minimizar (total-time)
- Minimizar (total-bateria-consumida)
- Minimizar métrica híbrida basada en suma de cuadrados

5.1. Variantes del problema y experimentos

Además del problema base, se han diseñado dos variantes. Una variante A donde los dos robots dependen el uno del otro para cumplir los objetivos establecidos con un claro cuello de botella, y una variante B donde ambos robots pueden actuar de forma paralela e independiente para cumplir con sus objetivos.

A continuación se mostrarán los resultados de los 3 problemas estudiados:

Caso Base

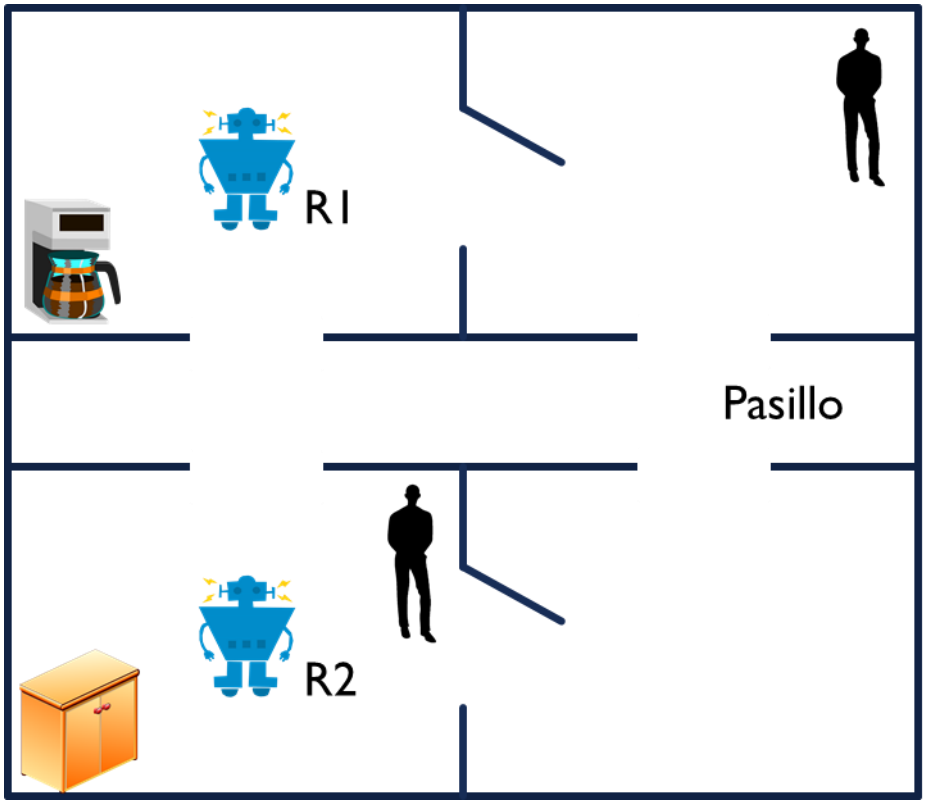


Figura 14: Distribución inicial del problema base

Planificador	Base	Temporal
Metric	17 pasos	-
LPG	15 pasos	15 pasos/ 27 total-time
Optic	15 pasos	18 pasos/ 39 total-time

	Min. Tiempo		Min. Recurso		Min. Cuadrados	
Planificador	**Recursos**	**Renovable**	**Recursos**	**Renovable**	**Recursos**	**Renovable**
LPG	15 pasos/ 17,5 total-time	17 pasos/ 24 total-time	24 bateria / 27 total-time	24 bateria / 30 total-time	39 bateria / 19,5 total-time (3042 score)	34 bateria / 27 total-time (4072 score)
Optic	18 pasos/ 29,5 total-time	17 pasos/ 27 total-time	34 bateria / 39 total-time	-	- No soportado	- No soportado

Variante A

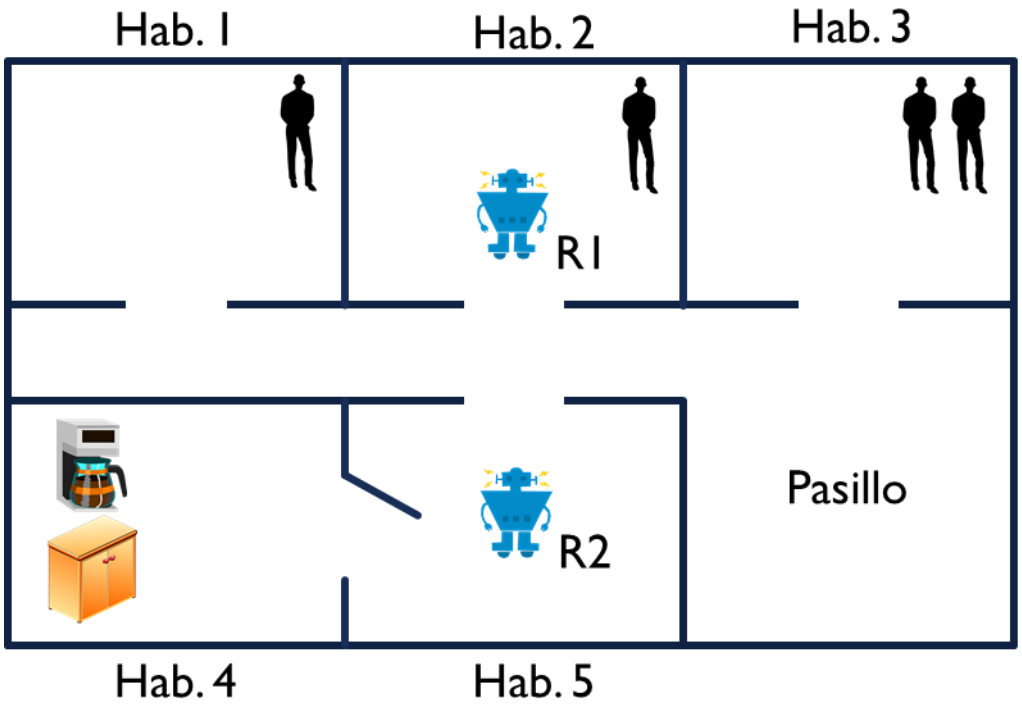


Figura 15: Distribución inicial del problema Variante A

Planificador	Base	Temporal
Metric	32 pasos	-
LPG	28 pasos	26 pasos/ 47 total-time
Optic	32 pasos	28 pasos/ 47 total-time

	Min. Tiempo		Min. Recurso		Min. Cuadrados	
Planificador	**Recursos**	**Renovable**	**Recursos**	**Renovable**	**Recursos**	**Renovable**
LPG	28 pasos/ 34 total-time	34 pasos/ 49,8 total-time	40 bateria / 45 total-time	42 bateria / 51,8 total-time	61 bateria / 34,5 total-time (8482 score)	54 bateria / 47,8 total-time (12055 score)
Optic	28 pasos/ 39,5 total-time	35 pasos/ 48,9 total-time	58 bateria / 41 total-time	-	- No soportado	- No soportado

Variante B

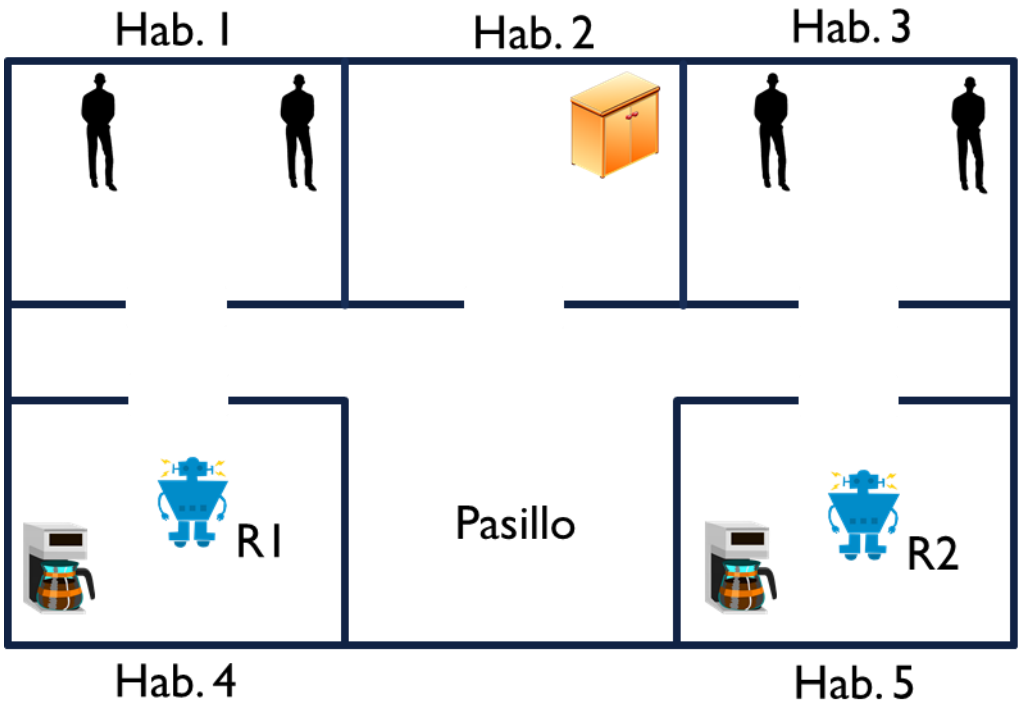


Figura 16: Distribución inicial del problema Variante B

Planificador	Base	Temporal
Metric	36 pasos	-
LPG	24 pasos	28 pasos/ 30 total-time
Optic	43 pasos	26 pasos/ 28 total-time

	Min. Tiempo		Min. Recurso		Min. Cuadrados	
Planificador	**Recursos**	**Renovable**	**Recursos**	**Renovable**	**Recursos**	**Renovable**
LPG	26 pasos/ 21 total-time	27 pasos/ 38 total-time	36 bateria/ 49 total-time	45 bateria/ 38,5 total-time	46 bateria/ 24 total-time (4420 score)	36 bateria/ 31 total-time (5140 score)
Optic	-	-	-	-	- No soportado	- No soportado

5.2. Ejemplo de plan

A continuación, mostramos un plan en el dominio de recursos renovables para el problema base, minimizando tiempo (17 pasos, 24 total-time):

```
0.0003:  (COGERTAZA R2 H3 T1 IZQ) [2.0000]
0.0003:  (COGERTAZA R2 H3 T2 DER) [2.0000]
0.0003:  (RECARGAR R1 H1) [3.0000]
3.0005:  (DESPLAZARROBOTTURBO R1 H1 PASILLO) [2.5000]
2.0008:  (DESPLAZARROBOT R2 H3 PASILLO) [2.0000]
5.5007:  (DARTAZAV R2 R1 PASILLO T2 DER IZQ) [1.0000]
5.5007:  (DARTAZAV R2 R1 PASILLO T1 IZQ DER) [1.0000]
6.5012:  (DESPLAZARROBOTTURBO R1 PASILLO H1) [2.5000]
9.0015:  (HACERTE R1 H1 T2 IZQ) [2.0000]
9.0015:  (HACERTE R1 H1 T1 DER) [2.0000]
11.0020: (RECARGAR R1 H1) [4.0000]
15.0022: (DESPLAZARROBOTTURBO R1 H1 PASILLO) [2.5000]
17.5025: (DARTAZALL R1 R2 PASILLO T2 IZQ DER) [2.0000]
19.5030: (DESPLAZARROBOTTURBO R1 PASILLO H2) [2.5000]
22.0032: (SERVIRTE R1 H2 P1 T1 DER) [2.0000]
19.5030: (DESPLAZARROBOT R2 PASILLO H3) [2.0000]
21.5032: (SERVIRTE R2 H3 P2 T2 DER) [2.0000]
```

De este plan cabe destacar tres elementos clave:

- La correcta paralelización de las acciones hechas por un mismo robot con distintas manos. (Resaltadas en naranja)
- La decisión del planificador de utilizar la acción **(:durative-action DesplazarRobot R2 H3 PASILLO)** en lugar de **(:durative-action DesplazarRobotTurbo R2 H3 PASILLO)** en el instante 2.0008 ya que esto no retrasaría el plan.
- La decisión del planificador de utilizar la acción **(:durative-action DesplazarRobot R2 PASILLO H3)** en lugar de **(:durative-action DesplazarRobotTurbo R2 PASILLO H3)** en el instante 19.5030 ya que esto no disminuiría la duración del plan.

5.3. Observaciones

Corrección de los dominios

Los dominios propuestos han mostrado ser correctos, ya que siempre se han podido extraer planes completos capaces de cumplir los objetivos establecidos.

Paralelización de las acciones

También se ha comprobado la paralelización de las acciones de los robots, aprovechando que cada uno dispone de dos manos.

Optimización de la solución

Además de producir soluciones correctas, el planteamiento de los dominios temporales y numéricos han mostrado ser optimizables por los distintos planificadores, como se puede observar en los resultados del apartado 5.1.

Preferencia de LPG sobre Optic

Con este dominio particular, parece que Optic causa problemas en cuanto a la optimalidad de la solución y la memoria utilizada, dando lugar a interrupciones de la ejecución sin devolver ningún resultado para instancias del problema más complejas como la variante B. Por lo tanto, podemos concluir empíricamente que LPG es un planificador más apropiado que Optic en esta implementación.

Métricas híbridas

El uso de una métrica híbrida para el dominio numérico ha dado buenos resultados, ofreciendo soluciones con más equilibrio entre la duración del plan y la batería total consumida. No obstante, el planificador Optic no es capaz de optimizar métricas no lineales, por lo cual debería usarse otra métrica distinta.

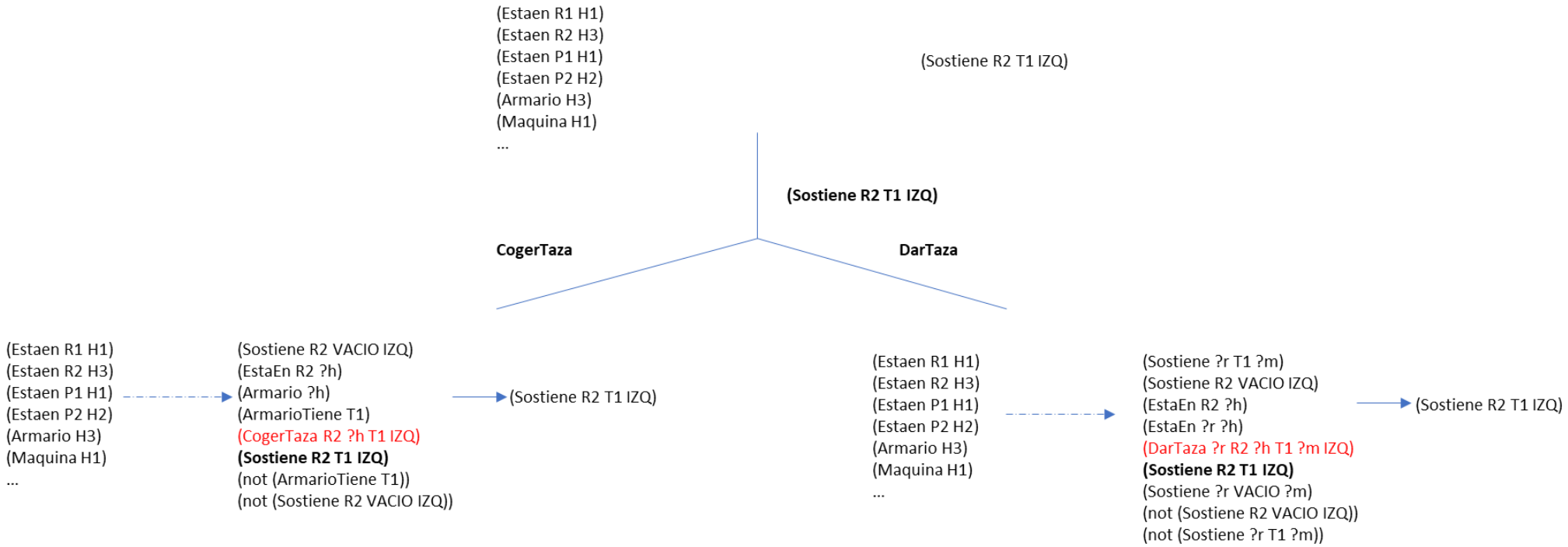
6. Partially-ordered Plan (POP)

En este ejercicio se ha construido el árbol de planificación de orden parcial (POP), haciendo 3 iteraciones del algoritmo y desarrollando sólo un nodo hijo a cada iteración. En este caso, se ha decidido un único objetivo a partir del cual se desarrollará este árbol:

(Sostiene R2 T1 IZQ)

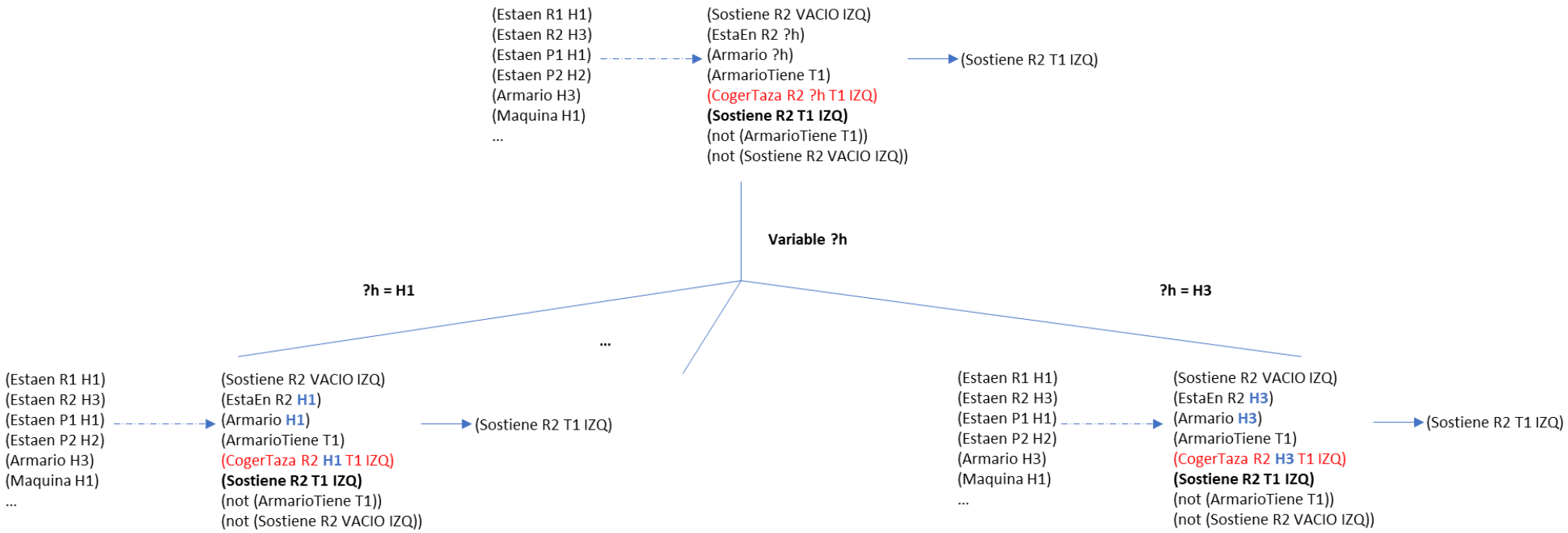
También se ha desarrollado previamente un árbol POP con el objetivo **(Servido P1)** pero debido a la simplicidad de este, se ha optado por el anterior predicado más intermedio para ilustrar la ramificación dominio.

Iteración 1 – Flaw: **Predicado (Sostiene R2 T1 IZQ)**



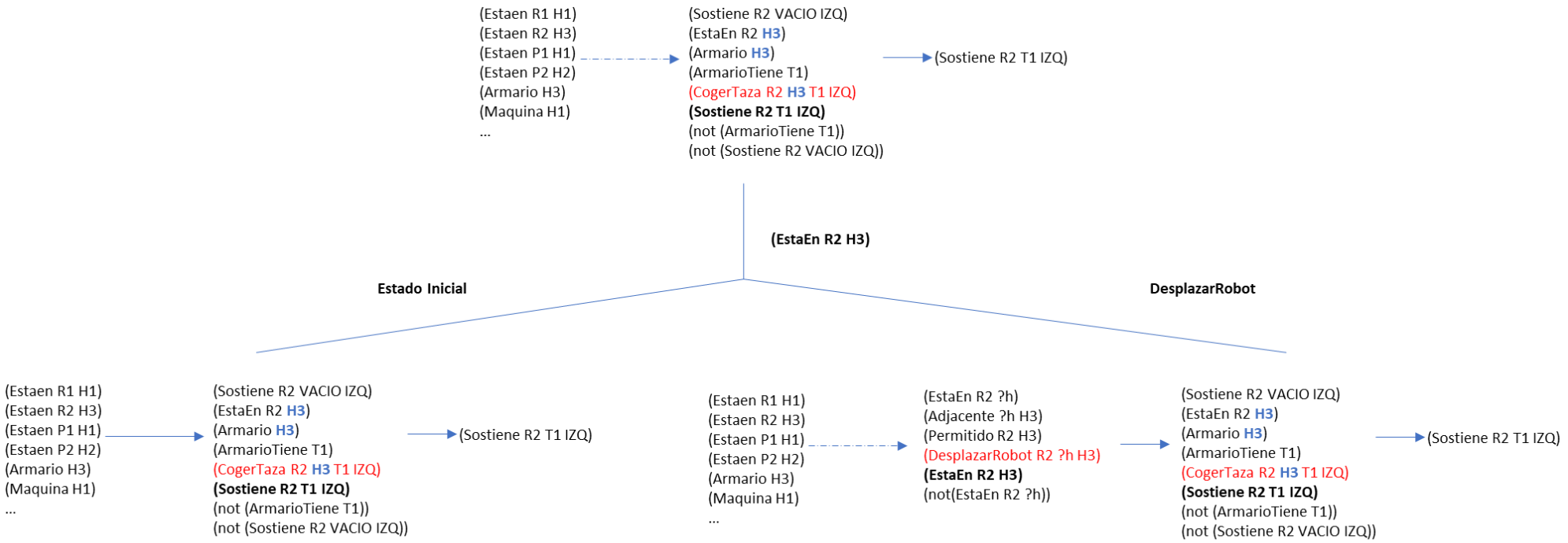
Desarrollamos el nodo hijo dado por “CogerTaza” en la iteración 2

Iteración 2 – Flaw: **Instanciación de la variable ?h**



Desarrollamos el nodo hijo dado por “?h = H3” en la iteración 3

Iteración 3 – Flaw: **Predicado (EstaEn R2 H3)**



6.1. Observaciones

Es fácil de ver que las dos acciones que modifican los predicados más clave en nuestro problema son **(:action DesplazarRobot …)** y **(:action DarTaza …)**, ya que anulan predicados como **(EstaEn …)** y **(Sostiene …)**. Si desarrollásemos el árbol POP más iteraciones, estas dos acciones serán las que den lugar a un mayor número de amenazas que resolver.

Por otro lado, las restricciones de acceso de los robots como **(Permitido …)** reducen el tamaño del problema de manera considerable, al no aparecer todos los predicados posibles en el Estado Inicial, y no existir ninguna acción que genere tal predicado.

El problema puede paralelizarse con dos planes parciales independientes para satisfacer **(Servido P1)** y **(Servido P2)**, una vez se desambigüen los robots R1 y R2 capaces de realizar la acción **(ServirTe …)** en la misma habitación que P1 y P2 dada la limitación **(Permitido …)** previamente comentada. Esto resultará en una ramificación en 2 sub-planes desde el estado final, uno para cada robot con el objetivo de servir una persona, entre los cuales no se deberían generar amenazas que resolver.

7. Graphplan

En este ejercicio se ha construido el grafo de planificación que puede encontrarse en el documento Excel “Graphplan” entregado junto a esta memoria. El grafo ha sido construido a partir del dominio base empleado en la primera sección de este trabajo, el cual consta de 5 habitaciones, 2 robots y 2 personas. A partir de dicho problema se ha generado el grafo de planificación relajado (sin tener en cuenta exclusiones mutuas y efectos negativos para las acciones) hasta alcanzar los 2 objetivos propuestos en el dominio. En base al grafo obtenido podemos responder a las siguientes preguntas:

7.1. Calcular el valor de las heurísticas “H_sum” y “H_max”

Dado que ambos objetivos, representados por los predicados “servido(P1)” y servido(P2)”, se alcanzan en el cuarto nivel del grafo de planificación, “H_sum” posee un valor de 8 al ser la suma de los niveles para cada objetivo; mientras que “H_max” tiene un valor de 4, pues independientemente del objetivo, el nivel máximo del grafo es 4.

7.2. Obtener un plan relajado

Durante la generación del grafo de planificación se han ido almacenando punteros para cada acción apuntando tanto a los predicados actuando como precondiciones para dicha acción, como a los predicados generados por la acción. Con esta estructura y partiendo de los predicados que representan los objetivos del dominio, es posible recorrer en orden inverso dichos punteros, obteniendo así una secuencia de acciones ordenadas en el tiempo que representan un plan relajado para la solución del problema.

El plan relajado obtenido es la secuencia de acciones:

- desplazarRobot(R1,H1,H2),
- desplazarRobot(R1,H1,PASILLO),
- desplazarRobot(R2,H3,PASILLO),
- cogerTaza(R2, H3, T1, IZQ/DER),
- cogerTaza(R2, H3, T2, IZQ/DER),
- darTaza(R2,R1,PASILLO,T1,DER/IZQ,DER/IZQ),
- darTaza(R2,R1,PASILLO,T2,DER/IZQ,DER/IZQ),
- hacerTe(R1,T1,IZQ/DER),
- hacerTe(R1,T2,IZQ/DER),
- darTaza(R1,R2,PASILLO,T2,DER/IZQ,DER/IZQ),
- servirTe(R1,H2,P1,T1),
- servirTe(R2,H2,P2,T2)

Estas acciones están marcadas en rojo en el documento Excel. Cabe destacar que en base a nuestra implementación del problema existen dos vías en el nivel final para alcanzar los objetivos: servir la taza 1 a la persona 1 y la taza 2 a la persona 2, o viceversa. Para obtener el plan relajado se ha empleado tan solo la primera de estas dos posibilidades.

Con esto obtenemos que el valor de la heurística del plan relajado es igual al número de acciones que componen el plan y por lo tanto es igual a 12.

7.3. ¿Cuál es la heurística más informada para este problema?

Una heurística “h_max” tiende a ofrecer el valor óptimo del plan en el caso de que todas las acciones que ocurren en un mismo nivel del GraphPlan se ejecutasen de forma paralela, es decir, “h_max” es la heurística que más paralelismo en las acciones supone. Por otra parte, “h_sum” mantiene parte de ese paralelismo, ya que no distingue acciones que ocurren en el mismo nivel del GraphPlan; pero también supone que existe cierta parte secuencial del plan dado que su resultado es la suma de los niveles en los que se alcanzan los distintos objetivos, y por lo tanto es la suma de las secuencias óptimas del plan relajado para alcanzar cada objetivo. Por último, la heurística de plan relajado solo contempla una ejecución secuencial de las acciones encontradas por el plan relajado ya que el valor de esta heurística es el número de acciones a ejecutar para alcanzar los objetivos.

Conociendo que algunas de las acciones necesarias para ejecutar el plan que resuelva el dominio proposicional del problema “Robots” sí que pueden ejecutarse en paralelo, pero otras no, la heurística mejor informada para este caso sería "h_sum”.