Universitat Politècnica de Catalunya

Inteligencia Artificial

Práctica Planificación

Jordi Muñoz, Jianing Xu, Yiqi Zheng

08/01/2024

Índices

1 Problema

El problema de esta práctica de planificación estaba relacionado con el tema de los libros de la práctica de SBC, donde se nos pedía realizar un SBC el cual recomendase novelas a un lector a partir de ciertos parámetros.

En esta práctica se nos proponía que una vez el lector ha recibido una serie de novelas a leer, realizáramos un planificador el cual elaborase un plan de lectura mensual en el que se le recomiende un orden de lectura, intentando balancear el número de páginas a leer en cada mes.

Para realizar el plan, teníamos que tener en cuenta que los libros, como en el mundo real, pueden tener uno o mas libros que se han de leer antes como en el caso de las sagas o series y por otra parte también pueden haber libros que se tengan que leer en paralelo como el caso de los crossovers o de los libros de universos como Star Wars.

Para resolver el problema, se nos pide que el sistema reciba los libros del catálogo, los libros predecesores a un libro los libros paralelos a un libro, los libros que el usuario ya ha leído en un pasado (en un año anterior) y los libros que el usuario quiere leer.

Por otra parte el resultado del problema debería dar información sobre qué libros tiene que leer el usuario, en qué mes ha de leerse dicho libro y para todos los libros incluidos en el plan, se deben respetar las relaciones de predecesores y paralelismos entre libros.

2 Dominio

En este apartado se va a realizar una descripción de las componentes que forman nuestro dominio. Estas son las variables, los predicados, las funciones fluentes y las acciones.

2.1 Tipos

- **libro:** tipo que representa un libro del catálogo. Es necesario para identificar qué objetos son libros y poder definir así, los libros del catálogo de un problema. Además, este tipo permite hacer una conjunción de objetos en los parámetros de los operadores más específica, y así, eficiente, entre otras operaciones más.
- **mes:** tipo que representa un mes del año. Su necesidad es similar a la anterior, para poder identificar qué objetos son meses del año.

2.2 Predicados

- **(quiere_leer ?l - libro):** indica si se quiere o se debe leer un libro ?l. Es necesario para identificar el objetivo del planificador, es decir, los libros que se quieren leer, y saber cuándo hemos llegado a una solución, básicamente que estos libros estén leídos o asignados para leer. Además, lo utilizamos para marcar como quiere_leer a esos libros que son predecesores y/o paralelos de un libro que se quiere leer, ya que se deben de leer según el enunciado del problema.
- **(leido ?l - libro):** indica si un libro ?l ya ha sido leído anteriormente o si se le ha asignado un mes para ser leído. Es necesario para marcar si le hemos asignado un mes a cada libro que se desea y se debe leer, y saber cuándo hemos llegado a una solución.
- **(es_predecesor ?antes - libro ?despues - libro):** indica si un libro ?antes es predecesor de un libro ?despues. Es necesario porque nos explicita las restricciones que supone ésta relación entre dos libros que nos será útil para realizar asignaciones (precondición de los operadores) y encontrar soluciones válidas, ya que si un libro ?antes es predecesor de un libro ?despues, para poder leer ?despues se debe haber leído ?antes en un mes anterior.
- **(es_paralelo ?l1 - libro ?l2 - libro):** Este predicado nos dice si un libro ?l1 es paralelo a un libro ?l2. El motivo es similar al de es_predecesor, nos explicita las restricciones que supone ésta relación

entre dos libros que nos será útil para realizar asignaciones (precondición de los operadores) y encontrar soluciones válidas, ya que si un libro ?l1 es paralelo de un libro ?l2, estos se deben leer en el mismo mes o en meses consecutivos.

2.3 Funciones fluentes

- **(paginas_disponibles_mes ?m - mes):** indica cuántas páginas aún se pueden leer en un mes ?m. Este valor disminuye a medida que se le asignan libros a ese mes ?m. Esta función fluente está hecha concretamente para la extensión 3 del planificador, donde se pide que se limite a 800 el número de páginas que se pueden leer en un mes.
- **(paginas ?l - libro):** indica el número de páginas que posee un libro. Siguiendo el caso anterior, este fluente es para la extensión 3 del libro que es necesario para saber y calcular cuántas páginas se van a leer en un mes dados unos libros asignados y así, no superar el límite de 800 páginas.
- **(numero_mes ?m - mes):** Este fluente sirve para poder asignarle un valor a los meses, permitiendo saber cuáles van antes que otros según el valor y posteriormente utilizarlo en las acciones. Si un valor de un mes m1 es más pequeño que el de otro mes m2, eso indica m1 es un mes anterior a m2.
- **(mes_leido ?l - libro):** Este fluente sirve como enlace entre las variables libro y mes, indicando en formato numérico, el mes en el que se ha leído un libro. Al principio todos los libros tienen este valor a 0, indicando que dentro de la ejecución del programa aún no se ha llegado a leer y no tiene mes asignado.

2.4 Acciones

- **transformar(?l1 - libro ?l2 - libro):** Esta acción recibe como parámetros 2 instancias de libros. Su función principal es comprobar dentro de la topología, que de los libros iniciales que se quiere leer, todos los que estén conectados a él de alguna forma, ya sea porque son sus predecesores o porque son sus paralelos, pasen a también quererse leer en caso de que no lo estuviesen previamente, ya que según el enunciado, se deberían de leer. El uso de este operador nos permite que únicamente asignemos para leer esos libros que se han marcado como querer leer (que sería también deber leer en nuestro caso), serviría como un pre-

procesamiento para el operador leer. Su justificación y funcionamiento se explica en la sección 3.1.1 Operador Transformar.

- **leer(?l - libro ?m - mes):** Esta acción recibe como parámetros 1 instancia de libro y 1 instancia de mes. Su función es asignar la lectura del libro al mes dados, siempre que se cumplan las condiciones necesarias para que ese libro pueda leerse en ese mes. Su justificación y funcionamiento se explica en la sección 3.1.2 Operador Leer.

3 Modelización

En este problema partimos de un estado inicial en el que se introducen una serie de libros al sistema de los que el usuario indicará cuales ya se ha leído y cuales quiere leer a parte de todas las relaciones que puedan haber entre ellos de predecesores y paralelos. Para modelizarlo en el archivo que representa el problema se pueden introducir tantos objetos del tipo libro, como libros hayan.

Con toda esta información, el algoritmo utilizará los operadores que explicaremos a continuación para llegar a un estado final el cual da como resultado la planificación indicando en que mes se debe leer cada libro.

3.1 Operadores

Para representar el problema, inicialmente pensamos en utilizar dos operadores, los cuales fuesen, leer_libro y quitar_libro, el primero se encargaría de ir añadiendo libros a la planificación teniendo en cuenta que todos las relaciones de predecesores y paralelos, y el segundo se encargaría de quitar libros a al planificación si en algún caso el algoritmo añadiese un libro que no tuviese en cuenta alguna relación de las que se especifican.

Tras hablar con nuestro profesor, entendimos mucho mejor como funcionaba el algoritmo que corre pddl ya que es una mezcla de el algoritmo A* con un greedy breadth-first el cual nos permite no tener un operador de quitar_libro ya que se encargará el propio algoritmo de ir recorriendo todas las ramas buscando, si es que la hay, una solución viable.

Más adelante, nos dimos cuenta de que las relaciones de predecesores y paralelos podían llevar a casos no triviales a la hora de hacer una planificación, ya que en el ejemplo de que el usuario informe de que quiere leer un libro, el cual tiene como predecesor a otro, sobre el cual no ha informado de forma explícita que lo quiera leer, en este caso en la planificación deberíamos también incluir el libro que precede al que el usuario quiere leer. Lo mismo pasa con las relaciones de paralelos. Por este motivo decidimos incluir un operador extra para solucionar estos casos el cual se llamaría transformar y se encargaría de hacer el proceso de indicar los libros que el usuario tiene que leer de forma implícita.

Por lo tanto finalmente nos quedamos con dos operadores, el de transformar y el de leer, que en un principio pensamos en dividirlos para tratar los libros que tienen algún predecesor de los libros que tienen algún paralelo, para que fuese todo más legible, pero más adelante vimos que no hacia falta hacer

esta división dado que como la precondición y los efectos de los operadores serían las mismos en todos los operadores y por lo tanto no nos aportaría nada beneficioso aparte de legibilidad.

A continuación explicamos los dos operadores:

3.1.1 Transformar

Este operador recibe como parámetros dos libros l1 y l2, y se encarga de mirar si hay alguna relación, ya sea que el que no se quiere leer sea predecesor del que se quiere leer o que l1 y l2 se tengan que leer en paralelo, tal que uno de los dos se encuentre en quiere_leer y el otro no y poner el que no se quiere_leer a quiere_leer.

3.1.2 Leer

Este operador recibe como parámetros un libro l y un mes m. Tiene como precondiciones, que el libro se quiera leer con el predicado quiere_leer, también, que el libro no se haya incluido ya en la planificación, con el predicado de leído; mirar si las páginas disponibles en el mes m son suficientes para que se pueda planificar el libro l al mes m, que todos los libros que se deban leer en paralelo a l ya tengan el predicado quiere_leer a true (para evitar que se planifique un libro que tenga alguna relación de paralelo con otro que aun no se ha especificado que se quiere leer) y como condición final, ver que se cumple todas las relaciones de l con otros libros que tenga como predecesores o en paralelo.

En el caso de que se cumplan todas las condiciones para el libro l y el mes m, entonces se marca l como leído, se indica que l se ha leído en el mes m y se decrementa el número de páginas que se pueden leer en el mes m con el número de páginas que tiene el libro l.

4 Desarrollo de modelos

Para resolver las distintas extensiones del problema propuesto, des de un primer momento desarrollamos un primer modelo que englobaba, como explicaremos a continuación, el nivel básico, la extensión 1 y la extensión 3, dejando las relaciones de paralelos que pedía la extensión 2 para un modelo posterior.

4.1 Primera versión

Lo primero de todo fue seguir la metodología que se nos enseñó en clase, empezar pensando las variables, los fluentes y los predicados que iríamos a utilizar, ya que así lo tendríamos claro a la hora de pensar las acciones que íbamos a implementar. Para llegar a algo decente, nos fue de mucha ayuda una lectura profunda del enunciado, preguntando todas las dudas que se nos ocurriesen al profesor de laboratorio. También fue de gran ayuda el hecho de asistir a las clases de teoría donde hicimos un par de problemas de pddl con explicación exhaustiva, en las que aprendimos distintas técnicas que aplicamos posteriormente a la práctica.

Una vez leída la práctica de primeras ya pensamos en esta primera versión, en la que agrupamos el nivel básico, la extensión 1 y la extensión 3. Hicimos esto ya que la fusión del nivel básico y la extensión 1 no comportaba un esfuerzo de grandes dimensiones y por otra parte, pensamos que añadir la extensión 3 sería una tarea muy sencilla si empezábamos enfocando el modelo hacía allí desde el primer momento.

4.2 Versión final

La versión final del planificador fue complicada de implementar. A parte de los predicados y fluentes que ya teníamos, añadimos un predicado es_paralelo para poder tratar esta característica adecuadamente. Teníamos que añadir el paralelismo entre libros, teniendo la dificultad de tener en cuenta de alguna manera los libros que son a la vez predecesores y paralelos a otro. Esto resultó en que tuviésemos bastantes versiones antes de llegar a la final, en la cual modificamos la acción de leer de tal manera que, tiene todo lo anterior y ahora además comprueba primeramente que todos los libros de tipo paralelo que se quieren leer y sus paralelos que no se quieren leer en un principio, pasen a quererse leer. Después de eso ya miramos si el libro es de tipo predecesor o paralelo, en caso de ser paralelo, hay que tener mucho cuidado con si es el primero de todos o no, porque en el caso que lo sea, si no se lee habrá un

bucle infinito y la planificación será irresoluble.

5 Juegos de prueba

Para hacer las pruebas, utilizamos el último modelo que hemos implementado, y como hay una gran parte del archivo del problema que es estática y siempre es la misma, no la incluimos en el documento ya que se puede ver en los propios archivos. Solo incluimos lo que varia y es necesario para probar los juegos de prueba pertinentes, la parte que varia al declarar los objetos y la parte que varía en el bloque *init* para inicializar los predicados de cada objeto correctamente. Hemos incluido también una representación de los problemas que vamos a evaluar con sus respectivos DAG, donde los nodos verdes son los libros que ya están leído previamente, los nodos amarillos son los que queremos leer y los nodos negros son los que no son de ningún tipo. Además, una arista de un nodo 1 que apunta al nodo 2 indica que 1 es un predecesor de 2. En cambio, si la arista es bidireccional, indica que 1 y 2 son paralelos.

5.1 Pruebas nivel básico

Para estas pruebas no utilizaremos relaciones de libros paralelos y los libros solo pueden tener 0 o 1 libro predecesor. Para probarlo, como no necesitamos páginas en los libros pondremos que todos los meses tienen capacidad 800 paginas y todos los libros tienen 800 páginas, así en cada mes solo se podrá leer un libro.

5.1.1 Prueba 1

Queremos probar que si un libro es predecesor de otro y todos se quieren leer, los predecesores se leen antes del que preceden.

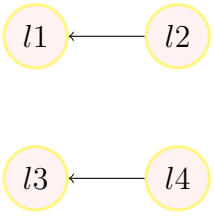

Figure 1: Representación gráfica del juego de prueba 1.1

Aquí el código del problema en pddl:

```
(:objects l1 l2 l3 l4 - libro)

(:init
```

```
    (quiere_leer l1)
    (quiere_leer l2)
    (quiere_leer l3)
    (quiere_leer l4)

    (es_predecesor l4 l3)
    (es_predecesor l2 l1)
)
```

Output del juego de prueba 1:

```
step     0: LEER L2 ENERO
         1: LEER L1 FEBRERO
         2: LEER L4 MARZO
         3: LEER L3 ABRIL
```

Vemos que como era de esperar, el libro l2 se lee antes que el libro l1, y el libro l4 se lee antes del libro l3.

5.1.2 Prueba 2

Queremos probar que si un libro, en este caso l8, no es predecesor de ningún libro, no se lea, que si un libro, en este caso l2, es predecesor de un libro que se quiere leer y no se indica que se quiere leer, se lea antes que el libro del cual es predecesor y finalmente que si un libro que ya se ha leído en el pasado y es predecesor de un libro que se quiere leer, en este caso l6, no se lea y por lo tanto l5 se pueda leer donde quiera.

Figure 2: Representación gráfica del juego de prueba 1.2

Aquí el código del problema en pddl:

```
(:objects l1 l2 l3 l4 l5 l6 l7 l8 - libro)

(:init
    (leido l6)

    (quiere_leer l1)
    (quiere_leer l3)
    (quiere_leer l5)
```

```
    (quiere_leer l7)

    (es_predecesor l2 l1)
    (es_predecesor l6 l5)
)
```

Output del juego de prueba 2:

```
step     0: LEER L5 ENERO
         1: LEER L3 FEBRERO
         2: LEER L7 MARZO
         3: TRANSFORMAR L1 L2
         4: LEER L2 ABRIL
         5: LEER L1 MAYO
         6: REACH-GOAL
```

Vemos que el plan es el esperado, l5 se lee al principio de todo puesto que no tiene restricción alguna, los libros que se quieren leer, que son l1, l3, l5 y l7, se acaban leyendo y por consecuencia de la relación que indica que l2 precede a l1, se aplica el operador de transformar y también se lee l2. Finalmente como l8 no se quiere leer y no es predecesor de ninguno que se quiere leer, no se incluye en el plan de lectura.

5.2 Pruebas extensión 1

Para estas pruebas no utilizaremos relaciones de libros paralelos y los libros pueden tener mas de un libro predecesor. Para probarlo, como no necesitamos páginas en los libros pondremos que todos los meses tienen capacidad 800 páginas y todos los libros tienen 0 páginas, así en cada mes se puede leer más de un libro, y se podrán ver mejor los solapamientos entre cadenas de predecesores.

5.2.1 Prueba 1

En esta prueba queremos probar que dadas dos cadenas de predecesores, $l1 \rightarrow l2 \rightarrow l3$ y $l4 \rightarrow l5 \rightarrow l6$, se respetan los predecesores y se permiten solapamientos entre ellas, es decir l1 podría leerse en el mismo mes que l4 ya que no hay ninguno que preceda al otro. También esperamos ver que se aplica algunas veces el operador transformar para indicar que l2, l3, l5 y l6 se quieren leer ya que no se hace de forma explícita.

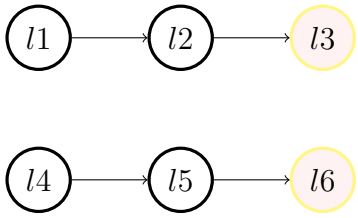

Figure 3: Representación gráfica del juego de prueba 2.1

Aquí el código del problema en pddl:

```
(:objects l1 l2 l3 l4 l5 l6 - libro)

(:init
    (quiere_leer l3)
    (quiere_leer l6)

    (es_predecesor l1 l2)
    (es_predecesor l2 l3)

    (es_predecesor l4 l5)
    (es_predecesor l5 l6)
)
```

Output del juego de prueba 1:

```
step     0: TRANSFORMAR L6 L5
         1: TRANSFORMAR L5 L4
         2: LEER L4 ENERO
         3: LEER L5 FEBRERO
         4: LEER L6 DICIEMBRE
         5: TRANSFORMAR L3 L2
         6: TRANSFORMAR L2 L1
         7: LEER L1 ENERO
         8: LEER L2 FEBRERO
         9: LEER L3 DICIEMBRE
        10: REACH-GOAL
```

Vemos que se cumple con lo esperado. Los libros predecesores de los que se quiere leer, son leídos. Además, l1 se lee en el mismo mes que l4 y se leen antes que l2 y l5, y estos se leen antes que l3 y l6, aplicándose el operador transformar las veces que hace falta y donde es debido, y cumpliendo la relación de predecesor.

5.2.2 Prueba 2

En esta prueba, vamos a utilizar una cadena más compleja de predecesores. Queremos observar que se respetan los predecesores, también, qué ocurre

cuando un libro que es predecesor ya está leído, qué ocurre cuando hay libros que no se quieren leer y tienen como predecesor uno que se quiere leer y vamos a permitir el solapamiento entre libros dentro de un mismo mes.

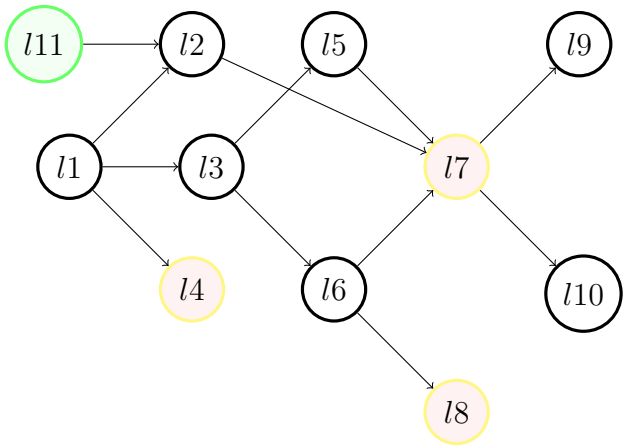

Figure 4: Representación gráfica del juego de prueba 2.2

Aquí el código del problema en pddl:

```
(:objects l1 l2 l3 l4 l5 l6 l7 l8 l9 l10 l11 - libro)

(:init
        (leido l11)
        (quiere_leer l4)
        (quiere_leer l7)
        (quiere_leer l8)

        (es_predecesor l11 l2)

        (es_predecesor l1 l2)
        (es_predecesor l1 l3)
        (es_predecesor l1 l4)

        (es_predecesor l2 l7)

        (es_predecesor l3 l5)
        (es_predecesor l3 l6)

        (es_predecesor l5 l7)

        (es_predecesor l6 l7)
        (es_predecesor l6 l8)

        (es_predecesor l7 l9)
```

```
        (es_predecesor l7 l10)
)
```

Output del juego de prueba 2:

```
step     0: TRANSFORMAR L7 L6
         1: TRANSFORMAR L6 L3
         2: TRANSFORMAR L7 L5
         3: TRANSFORMAR L7 L2
         4: TRANSFORMAR L2 L11
         5: TRANSFORMAR L4 L1
         6: LEER L1 ENERO
         7: LEER L2 FEBRERO
         8: LEER L3 FEBRERO
         9: LEER L4 DICIEMBRE
        10: LEER L5 MARZO
        11: LEER L6 MARZO
        12: LEER L7 DICIEMBRE
        13: LEER L8 DICIEMBRE
        14: REACH-GOAL
```

Vemos que el plan está bien hecho. Primero, transforma todos los predecesores en cadena a que se deben leer para posteriormente, empezar a leer los libros por el primero de todos los predecesores. Además, los libros l9, l10 y l1 no se llegan a leer porque, o ya están leídos, o son libros que no hace falta leerse.

5.3 Pruebas extensión 2

Para estas pruebas utilizaremos relaciones de libros paralelos y los libros pueden tener mas de un libro predecesor. Para probarlo, como no necesitamos páginas en los libros pondremos que todos los meses tienen capacidad 800 paginas y todos los libros tienen 800 páginas, así en cada mes solo se podrá leer un libro y se verá mejor como se respetan las relaciones de libros paralelos, puesto que si ponemos que tienen 0 páginas como en el caso anterior, se leeran todos en el mismo mes.

5.3.1 Prueba 1

Para este juego de prueba queremos probar que teniendo en cuenta que solo se puede leer un libro cada mes si tenemos que l1 y l2 se deben leer en paralelo y que l1 y l5 se quieren leer en paralelo, deberíamos ver que l1 se lee entre l2 y l5, lo mismo para l3, l4 y l6.

Aquí el código del problema en pddl:

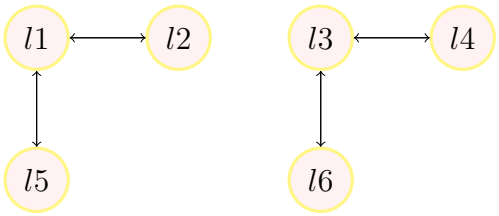

Figure 5: Representación gráfica del juego de prueba 3.1

```
(:objects l1 l2 l3 l4 l5 l6 - libro)

(:init
    (quiere_leer l1)
    (quiere_leer l2)
    (quiere_leer l3)
    (quiere_leer l4)
    (quiere_leer l5)
    (quiere_leer l6)

    (es_paralelo l1 l2)
    (es_paralelo l2 l1)

    (es_paralelo l3 l4)
    (es_paralelo l4 l3)

    (es_paralelo l5 l1)
    (es_paralelo l1 l5)

    (es_paralelo l6 l3)
    (es_paralelo l3 l6)
)
```

Output del juego de prueba 1:

```
step     0: LEER L2 ENERO
         1: LEER L1 FEBRERO
         2: LEER L5 MARZO
         3: LEER L4 ABRIL
         4: LEER L3 MAYO
         5: LEER L6 JUNIO
```

Vemos que se cumple con lo esperado dado que que l1 se lee un mes después de l2 y un mes antes que l5 y por tanto se respetan las reglas de paralelos entre ellos. Lo mismo pasa con los libros l3, l4 y l6.

5.3.2 Prueba 2

Para este juego de prueba queremos probar que teniendo en cuenta que solo se puede leer un libro cada mes si tenemos que l1 y l2, l2 y l4 y l4 y l5, se quieren leer en paralelo deberíamos ver que se recomienda leerlos en el mismo mes, un mes antes o un mes después, por otra parte l3 no haría falta leerlo puesto que ya lo hemos leído y l6 se debería leer antes que l1 y l5.

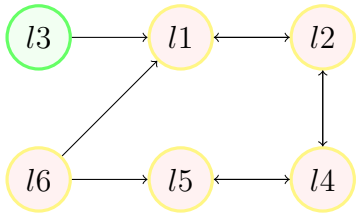

Figure 6: Representación gráfica del juego de prueba 3.2

Aquí el código del problema en pddl:

```
(:objects l1 l2 l3 l4 l5 l6 - libro)

(:init
    (leido l3)

    (quiere_leer l1)
    (quiere_leer l2)
    (quiere_leer l4)
    (quiere_leer l5)

    (es_paralelo l1 l2)
    (es_paralelo l2 l1)
    (es_paralelo l2 l4)
    (es_paralelo l4 l2)

    (es_predecesor l3 l1)

    (es_paralelo l4 l5)
    (es_paralelo l5 l4)

    (es_predecesor l6 l5)
    (es_predecesor l6 l1)

)
```

Output del juego de prueba 2:

```
step      0: LEER L2 MARZO
          1: LEER L4 ABRIL
```

```
2: TRANSFORMAR L5 L6
3: LEER L6 ENERO
4: LEER L1 FEBRERO
5: LEER L5 MAYO
6: TRANSFORMAR L1 L3
7: REACH-GOAL
```

Vemos que se cumple con lo esperado dado que se cumple que l1 y l2, l2 y l4 y l4 y l5, se leen en el mismo mes, un mes antes o un mes después, por otra parte l3 no se lee y l6 se lee antes que l1 y l5.

5.4 Pruebas extensión 3

Para estas pruebas ya empezaremos asignar un número de páginas diferentes de 0 u 800 a los diferentes libros combinándolo con las extensiones anteriores, conteniendo relaciones de predecesores y paralelos.

5.4.1 Prueba 1

En esta prueba, tenemos unas relaciones entre libros más compleja de predecesores y paralelos, el cuál el objetivo es leerse los libros l4, l7 y l8. Para conseguirlo, tendremos que leernos sus predecesores en un mes anterior a ellos, siempre cumpliendo las relaciones de paralelos, por ejemplo entre l5 y l6, o entre l4 y l7. Además, cada libro tendra un número de páginas de entre 0 a 800, que se tendrá que tener en cuenta cuando lo asignemos a un mes, ya que un mes cualquiera no puede contener libros que juntos superen las 800 páginas.

A continuación, mostramos una representación gráfica de las relaciones:

Seguidamente, el código del problema en pddl:

```
(:objects l1 l2 l3 l4 l5 l6 l7 l8 l9 l10 l11 - libro)

(:init
        (leido l11)
        (quiere_leer l4)
        (quiere_leer l7)
        (quiere_leer l8)

        (es_predecesor l11 l2)
        (es_predecesor l1 l2)
        (es_predecesor l1 l3)
        (es_predecesor l1 l4)
        (es_predecesor l2 l7)
        (es_predecesor l3 l5)
```

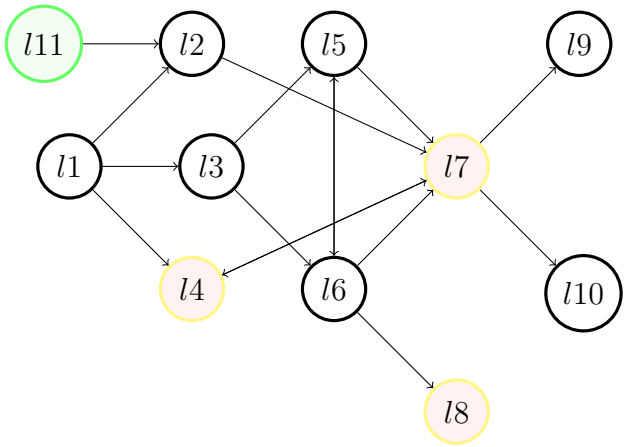

Figure 7: Representación gráfica del juego de prueba 4.1

```
        (es_predecesor l3 l6)
        (es_predecesor l5 l7)
        (es_predecesor l6 l7)
        (es_predecesor l6 l8)
        (es_predecesor l7 l9)
        (es_predecesor l7 l10)

        (es_paralelo l6 l5)
        (es_paralelo l5 l6)
        (es_paralelo l4 l7)
        (es_paralelo l7 l4)

        (= (paginas l1) 500)
        (= (paginas l2) 200)
        (= (paginas l3) 100)
        (= (paginas l4) 530)
        (= (paginas l5) 350)
        (= (paginas l6) 450)
        (= (paginas l7) 340)
        (= (paginas l8) 300)
        (= (paginas l9) 550)
        (= (paginas l10) 270)
        (= (paginas l11) 600)
)
```

Output del juego de prueba 1:

```
step    0: TRANSFORMAR L8 L6
        1: TRANSFORMAR L6 L3
        2: TRANSFORMAR L7 L5
        3: TRANSFORMAR L7 L2
        4: TRANSFORMAR L2 L11
```

```
 5: TRANSFORMAR L4 L1
 6: LEER L1 ENERO
 7: LEER L2 FEBRERO
 8: LEER L4 ABRIL
 9: LEER L3 FEBRERO
10: LEER L5 MARZO
11: LEER L6 MARZO
12: LEER L7 MAYO
13: LEER L8 MAYO
14: REACH-GOAL
```

Si analizamos la solución de la planificación, se asigna la lectura de los libros l4, l7 y l8, además de sus predecesores, que son leídos en meses anteriores.

Además, en las relaciones paralelas entre l4 (abril) y l7 (mayo), y entre l5 (marzo) y l6 (marzo) observamos que se cumple la condición de paralelo.

Si nos fijamos en la condición de no superar el número de páginas del mes, l4 y l7 aún y ser paralelos, no se les ha asignado el mismo mes porque 530 (l4) + 340 (l7) = 870 páginas que supera las páginas posibles en un mes. Sin embargo, l5 y l6 sí se les ha asignado el mismo mes ya que 350 (l5) + 450 (l6) = 800 páginas. Si hacemos el mismo procedimiento de sumar las páginas de los libros asignados a un mes, vemos que ninguno supera las 800 páginas.

Los libros que no se quieren ni se deberían leer, que son l9 y l10, no se les ha asignado su lectura, ni tampoco l11 porque ya está leído.

La planificación es correcta.

5.4.2 Prueba 2

En esta prueba, vamos a tener 3 secuencias de libros predecesores, añadiendo la dificultad de que 2 secuencias están enlazadas por paralelismos. También hacemos que la cantidad de páginas que tiene cada libro, sea una cantidad que haga que como mucho se puedan leer 2 libros por mes en caso que de una planificación válida. Vamos a intentar observar que se encuentre una planificación correcta, respetando el límite mensual de páginas y, las relaciones que hay entre los libros.

Aquí el código del problema en pddl:

```
(:objects l1 l2 l3 l4 l5 l6 l7 l8 l9 - libro)

(:init
        (= (paginas l1) 800)
        (= (paginas l2) 100)
        (= (paginas l3) 500)
```

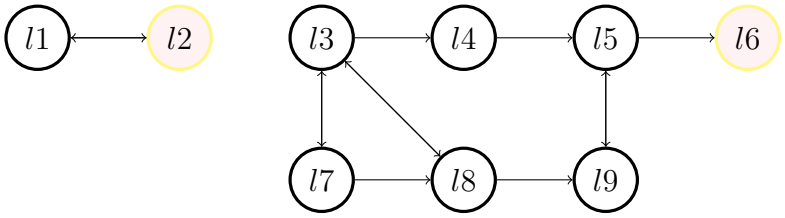

Figure 8: Representación gráfica del juego de prueba 4.2

```
        (= (paginas l4) 700)
        (= (paginas l5) 500)
        (= (paginas l6) 300)
        (= (paginas l7) 200)
        (= (paginas l8) 200)
        (= (paginas l9) 400)

        (quiere_leer l6)
        (quiere_leer l2)

        (es_predecesor l3 l4)
        (es_predecesor l4 l5)
        (es_predecesor l5 l6)

        (es_predecesor l7 l8)
        (es_predecesor l8 l9)

        (es_paralelo l1 l2)
        (es_paralelo l2 l1)

        (es_paralelo l3 l7)
        (es_paralelo l7 l3)

        (es_paralelo l3 l8)
        (es_paralelo l8 l3)

        (es_paralelo l5 l9)
        (es_paralelo l9 l5)
)
```

Output del juego de prueba 2:

```
step    0: TRANSFORMAR L6 L5
        1: TRANSFORMAR L5 L4
        2: TRANSFORMAR L4 L3
        3: TRANSFORMAR L3 L7
        4: LEER L7 AGOSTO
        5: TRANSFORMAR L3 L8
        6: LEER L3 AGOSTO
```

```
 7: LEER L4 OCTUBRE
 8: LEER L8 SEPTIEMBRE
 9: TRANSFORMAR L5 L9
10: LEER L5 NOVIEMBRE
11: LEER L6 DICIEMBRE
12: LEER L9 DICIEMBRE
13: TRANSFORMAR L2 L1
14: LEER L1 ENERO
15: LEER L2 FEBRERO
16: REACH-GOAL
```

Podemos observar que se encuentra una planificación correcta, ya que si sumamos la cantidad de páginas que se leen cada mes, no se superan las 800 establecidas; siendo estas 800 en enero, 100 en febrero, 700 en agosto, 200 en septiembre, 700 en octubre, 500 en noviembre y 700 en diciembre. La parte complicada se encuentra en la relación que hay entre los libros l3, l7 y l8, donde l3 es paralelo con l7 y l8, y a la vez l7 es un libro predecesor de l8. Se podría hacer de tal forma que l7 se leyese primero, luego l3 en el siguiente mes, y finalmente l8, pero el planificador se ha decantado por juntar l3 y l7 en agosto, y l8 en septiembre. Al final, se cumplen todas las relaciones de predecesor y paralelismo, junto al límite de páginas.

6 Conclusión

La realización de esta práctica nos ha llevado a tener una imagen mucho más amplia de como de útil puede ser esta herramienta para resolver problemas que nos podemos encontrar en nuestra vida cotidiana, en los que se parte de una serie de requisitos y se quiere llegar a un objetivo, realizando un proceso de modelización en el que primero se resuelve una parte del problema global y se van añadiendo características para terminar modelando el problema tratando todos y cada uno de los requisitos que tiene.

Hemos visto el gran potencial de esta técnica la cual definiendo un conjunto de tipos, implementando una serie de predicados y construyendo unos operadores, en un tiempo de ejecución mucho más que razonable y con una cantidad de código muy pequeña, se pueden resolver problemas muy complejos.