PARTE 2 -> Planificación

a) Describe el **dominio** (incluyendo predicados, acciones, etc...) usando PDDL. Da una explicación razonada de los elementos que has escogido.

Algunas cosas que se tiene en cuenta en la solución propuesta:

- Intentar minimizar el número de operadores y el factor de ramificación de los mismos, de forma que se reduzca la exploración de qué operadores son aplicables en cada momento.
- Evitar operadores con parámetros similares, de forma que la existencia de unos objetos u otros dirija la instanciación de operadores.
- Usar tipos en las variables para reducir en lo posible la cantidad de objetos que el planificador comprobará para cada parámetro del operador.
- Intentar evitar el uso de exists y forall en las precondiciones y efectos de los operadores, ya que tienen un impacto negativo en el tiempo de cómputo y, en la práctica, aumentan el factor de ramificación por la existencia de variables ocultas (variables a instanciar que no son parámetros) dentro del operador.
- Es más importante que la ejecución sea eficiente, aunque la representación sea más compleja (es decir, añadiremos predicados, operadores y tipos si eso puede facilitar la labor del planificador).

Ej.:

```
(define (domain banquete)
        (:requirements :adl :typing)
        (:types comensal silla mesa - object)
        (:predicates
                (necesita ?x - comensal ?y - comensal)
                (nosoporta ?x - comensal ?y - comensal)
                (pertenece_a ?sl - silla ?ms - mesa)
                (izquierda_de ?sizq - silla ?sder -silla)
                (colocado_en ?c - comensal ?sl - silla)
                (colocado_mesa ?c - comensal ?m - mesa)
                (colocado ?c - comensal)
                (asignada ?sl - silla)
        )
        (:action colocar_comensal
                :parameters (?c - comensal ?sl - silla ?ms - mesa ?sizq - silla ?sder - silla)
                :precondition   (and (not(colocado ?c)) (not(asignada ?sl))
                                (pertenece_a ?sl ?ms) (izquierda_de ?sizq ?sl)
                                (izquierda_de ?sl ?sder)
                                (forall (?c2 - comensal)
                                        (and
                                        (imply (necesita ?c ?c2)
                                                (or (colocado_en ?c2 ?sizq) (colocado_en ?c2
                                                ?sder) )
```

```
                                    )
                                    (imply (nosoporta ?c ?c2)
                                            (and (colocado ?c2) (not (colocado_mesa ?c2
                                            ?ms) )
                                    )
                                    )
                            ) ) )
        :effect (and (colocado ?c) (colocado_en ?c ?sl) (colocado_mesa ?c ?ms)
        (asignada ?sl))
        )
)
```

Razonamiento:

En esta solución se distingue entre tres tipos de objetos: los comensales, las mesas y las sillas. Modelamos la capacidad de cada mesa modelando las sillas y a que mesa pertenecen. Se han creado varios predicados:

- necesita, que nos sirve para modelar la restricción de que una persona necesita tener a otra a su lado.
- nosoporta, que nos sirve para modelar la restricción de que una persona no puede tener a otra en la misma mesa.
- pertenece_a, que nos dice que una silla pertenece a una cierta mesa.
- izquierda_de, que nos dice que una silla está a la izquierda de otra. Como toda silla tiene otra a su izquierda, con este predicado es suficiente para describir la posición de las sillas en la mesa. Tampoco nos hace falta añadir un predicado (derecha_de ?sder - silla ?sizq - silla), ya que se puede expresar fàcilmente con (izquierda_de ?sizq - silla ?sder - silla).
- colocado_en, que es el predicado que permite modelar la solución obtenida indicando que un comensal ya ha sido colocado en una cierta silla.
- colocado, que nos dice si un comensal ya ha sido colocado en alguna silla. Este predicado se añade como filtro en la acción colocar_comensal, haciendo que el planificador sólo unifique la variable ?c con comensales que aún no han sido colocados. También se usa en el fichero de problema (ver apartado siguiente) para expresar la condición objetivo: el plan acabará cuando todos los comensales están sentados (que es una condición más fácil de comprobar que mirar, para todo comensal, si existe una silla tal que el comensal está colocado en la silla).
- colocado_mesa, es un predicado que en principio puede parecer redundante pero no lo es. De no tener este predicado en el caso de tener una restricción de nosoporta, necesitaríamos un exists para añadir una variable sl2 que sería la silla en la que está

colocado el comensal non-grato, y mirariamos entonces que la silla no pertenezca a la mesa en la que queremos colocar al comensal.

- asignada, que nos dice que una silla ya tiene a un comensal colocado encima de ella. Podríamos usar el predicado colocado_en para saber si una silla tiene alguna persona sentada, pero de esa forma para saber si una silla está libre o no tendríamos que comprobar que, para todos los comensales en el modelo del problema no hay ninguno que esté colocado_en esa silla (y esto implica el uso de forall o de (not exists)). Por ello es más efectivo añadir el predicado asignada, que sabiendo la silla nos dice si está asignada o no, y podemos usarlo como filtro en la precondición de la acción.

El modelo tiene un único operador: colocar_comensal. Se ha optado por un único operador que lo controle todo, en vez de diferentes operadores que coloquen personas según si tienen o no diferentes restricciones. Esta segunda opción no es menos compleja, al contrario, es más fácil que se nos escape la colocación de una persona que tiene restricción de necesidad y de no soportar si entra en el operador inadecuado. Las dos primeras condiciones de la precondición filtran comensales ya colocados y sillas ya asignadas, la tercera sirve para saber la mesa a la que pertenece la silla sl (algo que necesitaremos para controlar las restricciones nosoporta) y las dos últimas condiciones nos permiten saber las sillas que están al lado (a izquierda y a derecha) de la silla sl (algo que necesitaremos para controlar las restricciones necesita). A continuación se coloca un forall para gestionar las posibles restricciones del comensal ?c que queremos colocar con otro comensal ?c2 (como un comensal puede tener cero, una o más de una de cada una de las restricciones este cuantificador no lo podemos evitar añadiendo parámetros a la acción). En el caso de que ?c no soporte a ?c2, comprobamos que ?c2 no esté en la misma mesa que ?c (gracias al predicado colocado_mesa, que como ya se ha explicado evita la necesidad de un exists). En el caso de que ?c necesite a ?c2 también se ha podido evitar el uso de exists para saber donde se sienta ?c2, ya que solo hay dos opciones posibles: la silla a la izquierda de sl y la silla a la derecha de sl, que obtenemos con la unificación de dos parámetros extra (sizq y sder) en la cuarta y quinta condición de la precondición. Podemos colocar estos dos parámetros extra, ya que para toda silla siempre existe una silla a su izquierda y una silla a su derecha, independientemente de si el usuario que queremos sentar tiene restricciones o no. Si se cumplen todas las condiciones de la precondición, el efecto es que el comensal ?c será colocado en la silla ?sl, marcando también que ?c ha sido colocado y que ?sl ha sido asignada, para filtrar así futuras unificaciones de esos objetos. Es importante remarcar que este modelo no solo permite modelar más o menos comensales y más o menos mesas, sino también más o menos sillas en cada mesa (podemos incluso modelar mesas en las que caben más sillas y otras en las que caben menos).

b) Describe este **problema** usando PDDL.

```
(define (problem banquete-2mesas-7sillas-6sillas-12comensales)
        (:domain banquete)
        (:objects Antonio Ainhoa Lola Lolo Esperanza Enrique Susana Manu Manolito
                Susanita Jesús Quico - comensal
                s1_1 s1_2 s1_3 s1_4 s1_5 s1_6 s1_7 s2_1 s2_2 s2_3 s2_4 s2_5 s2_6 - silla
                m1 m2 - mesa
        )
        (:init
                (necesita Antonio Ainhoa) (necesita Lola Lolo) (necesita Esperanza Enrique)
                (necesita Susana Manu) (necesita Manolito Manu)
                (necesita Susanita Susana) (nosoporta Ainhoa Susana)
                (nosoporta Quico Esperanza)
                (pertenece_a s1_1 m1) (pertenece_a s1_2 m1)
                (pertenece_a s1_3 m1) (pertenece_a s1_4 m1)
                (pertenece_a s1_5 m1) (pertenece_a s1_6 m1)
                (pertenece_a s1_7 m1)
                (izquierda_de s1_1 s1_2) (izquierda_de s1_2 s1_3)(izquierda_de s1_3 s1_4)
                (izquierda_de s1_4 s1_5) (izquierda_de s1_5 s1_6) (izquierda_de s1_6 s1_7)
                (izquierda_de s1_7 s1_1)
                (pertenece_a s2_1 m2) (pertenece_a s2_2 m2) (pertenece_a s2_3 m2)
                (pertenece_a s2_4 m2) (pertenece_a s2_5 m2) (pertenece_a s2_6 m2)
                (izquierda_de s2_1 s2_2) (izquierda_de s2_2 s2_3) (izquierda_de s2_3 s2_4)
                (izquierda_de s2_4 s2_5) (izquierda_de s2_5 s2_6) (izquierda_de s2_6 s2_1)
        )
        (:goal (forall (?c2 - comensal) (colocado ?c2)))
)
```

Se ha creado un objeto para cada comensal y cada mesa.

También hemos creado un objeto por cada una de las sillas (que luego asociamos a las mesas a las que pertenecen).

Los nombres intentan hacerlo más fácil de leer, pero al planificador le da lo mismo el nombre escogido para cada objeto.

El estado inicial indica las restricciones de necesidad/rechazo y luego describe la colocación de las sillas en mesas. En el caso de las relaciones necesita, como el enunciado nos dice que incluso en el caso de necesidad en los dos sentidos el planificador sólo considera uno de los sentidos, para todas las parejas que se mencionan en el enunciado hemos usado un único predicado necesita. En el caso de la familia, para asegurar que se les pone a todos juntos en la mesa, se ha optado por hacer que cada hijo necesite a uno de los padres y que uno de los padres necesite al otro.

El estado objetivo usa el predicado colocado (el plan acaba cuando todos los comensales han sido colocados en mesas). Se podría haber descrito listando los doce predicados colocado (uno por comensal), pero en este caso se ha optado por la expresión forall en adl, ya que nos permite cambiar el número de comensales del problema sin tener que cambiar la descripción del estado objetivo, y no nos añade complejidad (en Fast Forward el forall en los objetivos se instancia una sola vez generando los predicados individuales, con un coste lineal respecto al número de objetos a explorar).