PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
Facultad de Ingeniería
IIC2613 Inteligencia Artificial
2022-2

Tarea 1

Aspectos generales

Formato y plazo de entrega

El formato de entrega son archivos con extensión .lp con un PDF para las respuestas teóricas. El lugar de entrega es en el repositorio de la tarea, en la *branch* por defecto, hasta el **miércoles 21 de septiembre a las 23:59 hrs.** Para crear tu repositorio, debes entrar en el enlace del anuncio de la tarea en Canvas.

Integridad Académica

Este curso se adhiere al Código de Honor establecido por la universidad, el cual tienes el deber de conocer como estudiante. Todo el trabajo hecho en esta tarea debe ser **totalmente individual**. La idea es que te des el tiempo de aprender estos conceptos fundamentales, tanto para el curso, como para tu formación profesional. Las dudas se deben hacer exclusivamente al cuerpo docente a través de las *issues* en GitHub.

Por otra parte, sabemos que estás utilizando material hecho por otras personas, por lo que es importante reconocerlo de la forma apropiada. Todo lo que obtengas de internet debes citarlo de forma correcta (ya sea en APA, ICONTEC o IEEE). Cualquier falta a la ética y/o a la integridad académica será sancionada con la reprobación del curso y los antecedentes serán entregados a la Dirección de Pregrado.

Comentarios adicionales

El objetivo de esta tarea es que puedan desarrollar la capacidad de modelar problemas a partir del lenguaje natural y resolverlos utilizando ASP en Clingo. Es fundamental que pongan énfasis en las justificaciones de sus respuestas, cuidando la redacción, ortografía; manteniendo el código ordenado y comentado. Aquellas respuestas que solo presenten resultados o código (sin contexto ni comentarios) no serán consideradas, mientras que tareas desordenadas pueden ser objeto de descuentos.

1. DCCajotier (2 pts.)

Estás paseándote por el DCC cuando ves un afiche que dice *"Se buscan alumnos para un proyecto secreto"*. Intrigado, sigues leyendo y te enteras de que se trata de un proyecto de software en el cual participan 5 personas y buscan un último afortunado integrante de este equipo sobre-remunerado. Para entrar al equipo debes resolver el siguiente acertijo acerca del equipo:

1. Hay 5 personas programando, sentados en fila.
2. La persona de Talca programa en papel.
3. La persona de Valparaíso escucha *lofi hip hop radio - beats to relax/study to.*
4. Quien programa en Python lo hace en un ENIAC [1].
5. La persona de Antofagasta programa en Javascript.
6. La persona que usa un ENIAC está inmediatamente a la derecha (tu derecha) de la persona que programa en tablet.
7. Quien bebe mate, escucha Cumbia villera.
8. Quien toma café programa en un Mac.
9. La persona sentada justo al medio programa en C#.
10. La persona de Santiago está sentada en el primer puesto.
11. La persona que toma RedBull está sentada al lado de la persona que escucha Jazz.
12. Quien bebe café está al lado de quien escucha Rock latino.
13. La persona que bebe Gatorade programa en Ruby.
14. La persona de Concepción bebe agua.
15. La persona de Santiago está al lado de quien programa con un celular Nokia.

Por temas de claridad es importante mencionar que todas las personas son de ciudades diferentes, programan en dispositivos diferentes y lenguajes diferentes, beben líquidos diferentes y escuchan música diferente. Se puede asumir que todos los lenguajes son compatibles en todos los dispositivos (Es posible programar Ruby en un ENIAC)

Tu misión es descubrir quién programa en Clingo (lenguaje restante), y quién escucha música de ascensor (la música restante).

[1] https://es.wikipedia.org/wiki/ENIAC

Figura 1: Dispocisión del equipo de programación

1.1. Actividad 1: Resolviendo con Clingo

Intentaste resolver el acertijo en papel y te diste cuenta de que era muy complejo. Es por esto que decides hacerlo en Clingo. En esta actividad debes escribir cada pista del acertijo en Clingo y llegar a la respuesta. Para que sólo te preocupes de escribir las pistas en Clingo, se te entregarán todos los átomos que necesites. También habrán comentarios en el código base para que sepas donde escribir cada pista. **No debes borrarlos** ya que se utilizarán al momento de evaluarte.

Debes crear el predicado `combinacion(P, C, D, L, B, M)` donde P es el número que representa a un programador, C corresponde a la ciudad, D a un dispositivo, L el lenguaje de programación, B el bebestible, y M el género musical. Tu programa debe entrega un modelo con todas instancias de `combinacion(P, C, D, L, B, M)` que satisfagan las pistas del acertijo.

El formato de entrega para esta parte es sólo el código base con las pistas del acertijo. No debes cambiarle el nombre al archivo ni dejarlo fuera de la carpeta en la que se encuentra originalmente.

2. DCCursos (2pts.)

Ya resolviste el puzzle con la ayuda de Clingo y por ser el primero en descifrarlo te has ganado el puesto en el proyecto misterioso. Estás sentado en la sala J Pinto del DCC y te revelan la misión. Debes crear un nuevo Planner para los alumnos y alumnas de ingeniería. Por lo eficiente que es (y por el cariño que le tienes), le dices al equipo que planeas utilizar Clingo para hacerlo. El equipo, encantado, acepta tu propuesta y se compromete crear los tests de prueba y un visualizador para que tengas mayor claridad del problema. Tu sólo debes encargarte de la lógica de la solución creando reglas en Clingo.

Los tests que se te entregarán son programas de Clingo que se deben correr antes de tu programa. Estos podrían tener los siguientes predicados:

- `semester(1..N)`: Representa la cantidad de semestres que se deben planificar. La sintáxis `(1..N)` es lo mismo que decir:

  ```
  semester(1).
  semester(2).
  semester(3).
  ...
  semester(N).
  ```

  En los tests verás que el predicado se muestra como `semester(1..bound)` donde `bound` es una variable que representa la cantidad de semestres a asignar. La variable se fija en el comando que se presenta más adelante en el enunciado.

- `course(C)`: Representa la existencia de un curso C.

- `prerequisite(C1, C2)`: Indica que el ramo C1 es prerrequisito del ramo C2. Esto quiere decir que el ramo C1 debe planificarse en un semestre previo al semestre en que se toma C2.

- `corequisite(C1, C2)`: Indica que el ramo C1 es correquisito del ramo C2. Esto quiere decir que C1 es un prerrequisito de C2 que se puede planificar en el mismo semestre que se toma C2.

- `dictates_on(C, P)`: Indica que el ramo C se dicta el primer semestre de cada año cuando P es 1. Indica que el curso C se dicta en el segundo semestre de cada año cuando P es 2.

- `approved_at(C, S)`: Indica que el curso C fue aprobado en el semestre S.

Un ejemplo de un test de prueba es el siguiente:

```
course(calculo1). % calculo1 es un curso.
course(quim).
course(lineal).
course(calculo2).
course(dinamica)

prerequisite(calculo1, calculo2). % calculo1 es prerrequisito de calculo2.
corequisite(lineal, dinamica). % lineal es correquisito de dinamica.

dictates_on(calculo1, 1). % calculo1 se dicta semestres impares.
dictates_on(calculo1, 2). % calculo2 tambien se dicta en semestres pares.
dictates_on(lineal, 1).
dictates_on(lineal, 2).
dictates_on(quim, 1).
dictates_on(quim, 2).
dictates_on(dinamica, 2). % dinamica solo se dicta en semestres pares.
```

```
    dictates_on(calculo2, 1).
    dictates_on(calculo2, 2).

    approved_at(calculo1, 1). % calculo1 fue aprobado en el semestre 1.
    approved_at(quim, 1).

    semester(1..bound). % Hay semestres del 1 al bound para plainificar los cursos.
```

Para probar tu código con los átomos del test debes correr el siguiente comando en tu terminal:

```
clingo -n 1 base.lp tests/testX.lp -c bound=X
```

Para poder utilizar el visualizador, y también para evaluar tu programa, es obligatorio que crees el predicado `course_on(C, S)` que indica que el curso C se debe tomar en el semestre S. El resto de los predicados que deben mostrar tus modelos se presentan en el siguiente trozo de código:

```
#show course_on/2.
#show prerequisite/2.
#show corequisite/2.
#show dictates_on/2.
#show approved_at/2.
```

Para usar el visualizador es necesario que corras tu programa con el siguiente comando:

```
clingo -n 1 base.lp tests/testX.lp -c bound=X | py process.py -o "output.json"
```

Luego, debes abrir el archivo `planner.html` y cargarle `output.json`. Una vez hecho esto se debería mostrar una visualización del output de tu programa. La figura 2 muestra un ejemplo del visualizador.

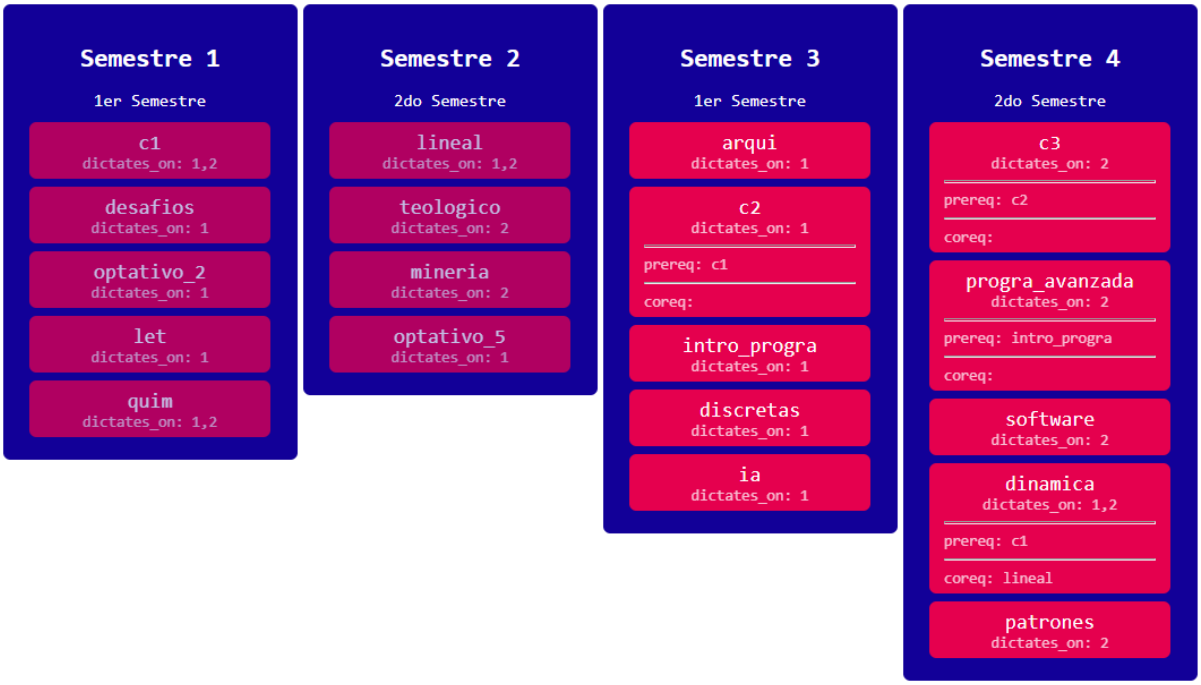


Figura 2: Visualizador de Planner

2.1. Actividad 1: Reglas Básicas (1pt.)

Lo primero que debes hacer es implementar las reglas más básicas que necesitará tu programa para tomar forma. Estas son las siguientes:

1. Los cursos aprobados no pueden tener un semestre asignado.
2. Todos los cursos que no estén aprobados deben tener un único semestre asignado.
3. Los semestres no pueden tener más de 5 ramos asignados.
4. Los cursos que sólo se dictan en semestres impares no pueden estar asignados a semestres pares y los cursos que sólo se dictan en semestres pares no pueden ser asignados a semestres impares.

2.2. Actividad 2: Ramos con requisitos (1pt.)

Existen ramos con prerrequisitos. Estas son las reglas que harán que tu planner esté completo:

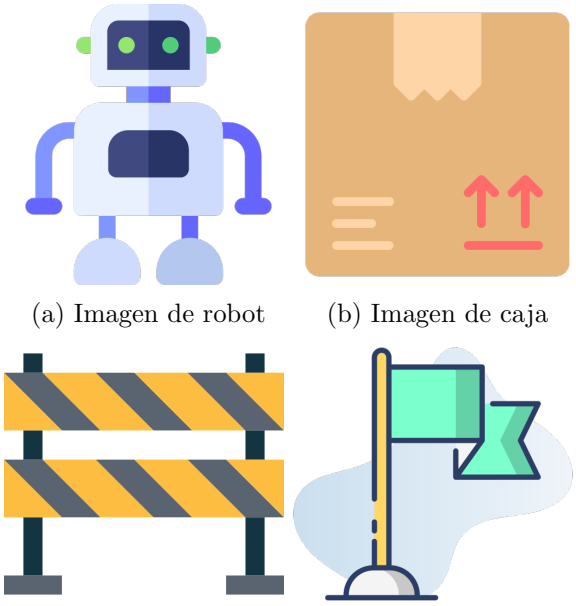
1. Si un ramo A es prerrequisito de un ramo B, entonces el ramo A no puede ser asignado en un semestre igual o posterior al del ramo B.
2. Si un ramo A es correquisito de un ramo B, entonces el ramo A no puede ser asignado en un semestre posterior al del ramo B.

El formato de entrega para esta parte es un archivo llamado `solution.lp` con comentarios explicando lo que hace cada regla. También debes crear un PDF donde expliques tu planteamiento, dando detalles de la lógica que utilizaste para dar solución al problema.

3. DCCharcos (2 pts.)

Hace un tiempo en el DCC se creó un programa que permitía que robots tomaran cajas y las dejaran en su estante correspondiente. Este programa consiste de un mapa con obstáculos, robots y cajas. En el código actual:

- Un robot puede desplazarse en las direcciones de: arriba, abajo, izquierda y derecha.
- Un robot puede esperar en una posición por un tiempo.
- Un robot es capaz de levantar una caja, únicamente si está adyacente a él. Además la caja pasa a estar en la posición del robot y este último mantiene su localización.
- Un robot ejecuta una acción en cada tiempo.
- Una caja puede ser llevada únicamente por un robot.
- Un robot no puede llevar más de una caja al mismo tiempo.
- Si hay una caja en el suelo, un robot no puede pasar por encima de ella.
- No puede haber más de un robot en la misma casilla.
- Un robot no puede pasar por un obstáculo.
- Luego de un tiempo *bound*, cada caja está en alguna de las estanterías/objetivos del mapa.



(a) Imagen de robot (b) Imagen de caja

(c) Imagen de obstáculo (d) Imagen de objetivo

Figura 3: Componentes originales

Los predicados más importantes que modelan este programa son:

```
rangeX(0..X). % Determina dimension en eje X
rangeY(0..Y) % Determina dimension en eje Y
robot(R). % Indica la presencia de un robot
box(B). % Indica la presencia de una caja
```

```
goal(X,Y) % Expone que hay una posicion objetivo en X,Y
obstacle(X,Y) % Expone que hay un obstaculo en X,Y
time(1..bound). % Define los tiempos T
robotOn(R,X,Y,T). % Expresa que un robot R esta en X,Y en el tiempo T
boxOn(B,X,Y,Z,T). % Expresa que una caja B esta en X,Y en T y si est  levantada (Z=1) o no
    (Z=0).
action(A). % Con A = up, down, left, right, wait, liftbox
exec(R,A,T). % Indica que un robot R ejecuta la accion A en T.
boxAdjacentToRobot(B, R, T). % Indica que hay una caja B adyacente a un robot R en T.
boxesAdjactentToRobot(R, T, N). % Expresa que hay N cajas adyacentes a un robot R en T.
boxPickedUp(B,T). % Indica que la caja B esta levantada en el tiempo T
robotLiftBox(R, B, T). % Indica que el robot R decide levantar la caja B en el tiempo T
at_goal(B, T) :- Expone que una caja esta en un objetivo en el tiempo T.
```

Para otro proyecto secreto del DCC se necesita extender el código y por tus increíbles habilidades en Clingo te piden a tí que los ayudes.

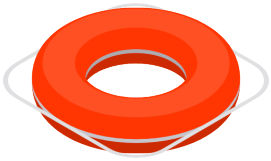
3.1. Actividad 1: Charcos (1pt.)

Ha sido un invierno extraño y ha habido más lluvia de lo esperado. Es por esto que se han creado goteras, y posteriormente charcos, a lo largo de la bodega. Tu misión es modelar una solución que contemple las complejidades de la lluvia.

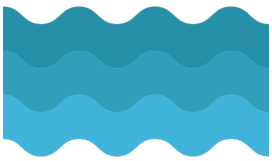
Para esto se te entregarán como inputs (además de los mencionados anteriormente) el predicado `water(X, Y)` que indica que en la posición `(X, Y)` hay un charco por el cual el robot no puede pasar. Para que el robot pueda atravesar los charcos, habrán disponibles flotadores a lo largo del mapa. La existencia de un flotador se representa con el predicado `float(F)`, mientras que el predicado `floatOn(F, X, Y, Z, T)` señala que un flotador F esta en X,Y en T y si está levantado (Z=1) o no (Z=0).

En esta actividad deberás:

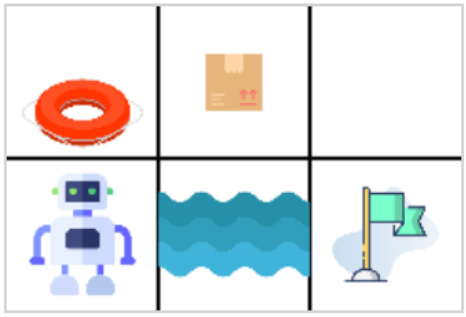
- Crear una acción para que un robot se ponga un flotador.
- Permitir que un robot se ponga un flotador únicamente si hay, al menos, uno adyacente a su posición.
- Una vez un robot tiene un flotador, ambos se deben mover de la misma forma.
- Un flotador puede ser llevado únicamente por un robot.
- Si hay un flotador en el suelo, un robot no puede pasar sobre él.
- Un robot no puede llevar más de un flotador al mismo tiempo.
- Un robot no puede pasar por una casilla de charco (*water*) a menos que tenga un flotador puesto.



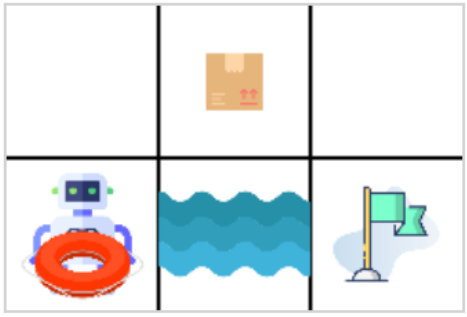
(a) Imagen de flotador



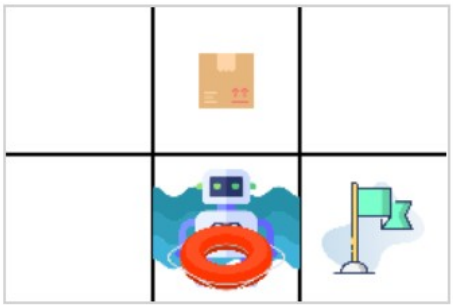
(b) Imagen de charco



(c) Ejemplo de robot en T



(d) Ejemplo de robot con flotador en $T+1$



(e) Ejemplo de robot con flotador en agua en $T+2$

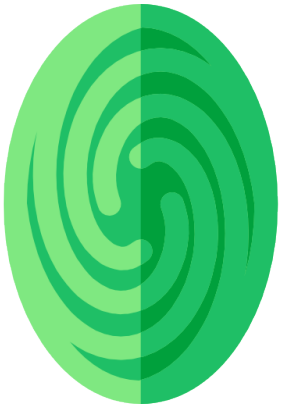
Figura 4: Ejemplo de uso válido de flotador

3.2. Actividad 2: Portales (1pt.)

Otra razón por la cual es necesario extender el código, es porque la escuela está trabajando con tecnología de punta: **portales unidireccionales**. Se te entregará la posición de los portales y la posición donde un robot termina al entrar en uno. Esto se representa con el predicado `portal(X1, Y1, X2, Y2)`, que indica que el portal está en la posición `(X1, Y1)` y el robot que lo cruce terminará en la posición `(X2, Y2)`. Tu misión es modelar el comportamiento de los portales y su interacción con los robots de modo que se

respete el cambio de posición una vez un robot ingrese a un portal.

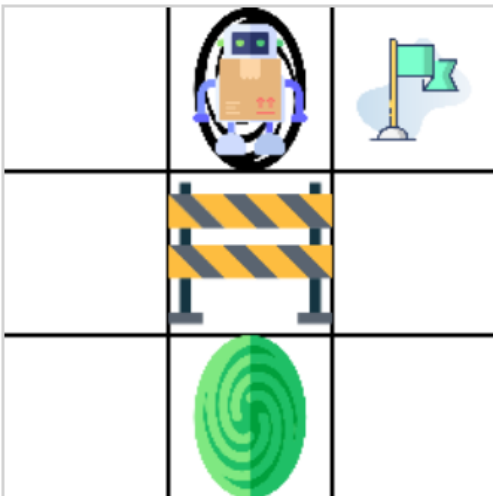
Nota: para realizar esto deberás agregar código, pero también **modificar el código base** que se te ha entregado.



(a) Imagen de entrada de portal

(b) Imagen de salida de portal

(c) Ejemplo de robot con portal en T



(d) Ejemplo de robot con portal en $T+1$

Figura 5: Ejemplo de viaje por portal(1,2,1,0)

3.3. Visualizador

Para esta parte de la tarea dispondrás de un visualizador temporal de la planificación que entregue tu programa.

Este requiere de la carpeta imgs que contiene las imágenes para identificar cada elemento en la bodega. Para hacer uso de esta funcionalidad deberás:

1. Mostrar los siguientes predicados en tu archivo de solución:

```
#show time/1.
#show robotOn/4.
#show boxOn/5.
#show obstacle/2.
#show rangeX/1.
#show rangeY/1.
#show goal/2.
#show water/2.
#show floatOn/5.
#show portal/4.
```

2. Correr los test de la forma:

```
clingo robots.lp tests/testX.lp -c bound=X | python process.py
```

Al hacer esto se generará un archivo llamado output.txt con el modelo que resulte. El visualizador corresponde al archivo llamado robot.html el cual podrás abrir en tu navegador favorito y cargar tu archivo output.txt.

El formato de entrega para esta parte es el mismo código base con comentarios explicando lo que hace cada regla nueva. También debes crear un PDF donde expliques tu planteamiento, dando detalles de la lógica que utilizaste para dar solución al problema.