PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
Facultad de Ingeniería
IIC2613 Inteligencia Artificial
2023-2

Tarea 2

Programación Lógica para el Razonamiento

Fecha de entrega: Sábado 9 de septiembre a las 23:59 hrs

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 sábado 9 de septiembre a las 23:59 hrs. Para crear tu repositorio, debes entrar en el enlace del anuncio de la tarea en Canvas. Por último, recuerda que los cupones de atraso son días **no hábiles** extra.

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.

Se recomienda fuertemente apoyarse de los apuntes del curso para el desarrollo de la tarea. En particular el uso de Clingo y modelación de problemas.

1. Reflexión y Teoría (2 pts.)

Mucha gente argumenta que el podcast de Lex Fridman en YouTube es imperdible para todo quien se dedique a la inteligencia artificial. En mayo de este año, Lex publicó una entrevista a Stephen Wolfram, creador del answer engine Wolfram Alpha, que permite responder consultas lógicas a partir de información externa. Para contestar esta pregunta, primero ve la sección sobre la naturaleza de la verdad en el video `https://www.youtube.com/watch?v=PdE-waSx-d8&t=7767s` (2:09:27 - 2:30:49).

1. En la entrevista, Lex propone que un modelo de lenguaje natural podría acercarse a tener una respuesta correcta a algunas preguntas subjetivas con suficiente información para su entrenamiento (por ejemplo, ¿es X persona buena?). Wolfram está en desacuerdo y plantea que la existencia de la verdad requiere de reglas o requisitos lógicos que la sustenten. ¿Estás de acuerdo con alguno de ellos? ¿Por qué? Explica tu punto de vista dentro de la discusión (puede ser el mismo que el de alguno de los oradores).

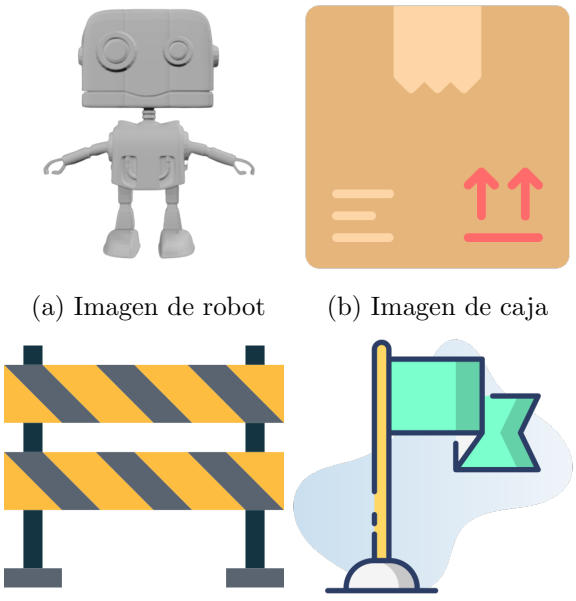
2. ¿Crees que el concepto de verdad puede cambiar a lo largo de la historia? ¿Varía la verdad entre distintos contextos culturales? Da tres ejemplos que respalden tu respuesta.

3. Más adelante en la entrevista, Wolfram habla sobre el uso de LLMs como interfaces de comunicación y su uso para comunicar hechos (y ejemplifica con pedir un permiso para extraer peces), ¿cómo el uso de herramientas de lenguaje natural por uno o ambos extremos de la comunicación podrían significar la pérdida o cambios de información relevante e impactar a esta? Da tres ejemplos de escenarios con sus respectivas consecuencias.

2. DCCarrybot (2 pts.)

Hace un tiempo en el DCC se creó un programa controlador de robots para que éstos tomaran cajas y las dejaran en su estante correspondiente. Este programa mantiene 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 1: Componentes originales

Los predicados más importantes que modelan este programa son:

```
rangeX(0..X).       % Determina dimension del mapa en eje X
rangeY(0..Y).       % Determina dimension del mapa en eje Y
robot(R).           % Indica la existencia de un robot
box(B).             % Indica la existencia de una caja
goal(X,Y).          % Declara que hay una posicion objetivo en X,Y
obstacle(X,Y).      % Declara 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 está 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.

2.1. Actividad 1: Direccionalidad de Robots

En esta nueva versión, los robots tienen asociada una dirección de movimiento. Esto significa que al avanzar, se desplazan a la casilla que están "mirando". Además de poder avanzar, los robots pueden realizar giros de 90 grados a la izquierda o a la derecha. Por ejemplo, si un robot está mirando hacia arriba y necesita mirar hacia abajo, podría realizar dos giros, ambos a la derecha o ambos a la izquierda. Cambiar de dirección se considera como una acción. Deberás:

1. Modelar la dirección de cada robot a través de un parámetro adicional de dirección para el predicado `robotOn` que toma el valor de:
   - 1 si está mirando hacia arriba
   - 2 si está mirando hacia abajo
   - 3 si está mirando hacia la izquierda
   - 4 si está mirando a la derecha

   La dirección debe encontrarse en el último parámetro de robotOn:

   ```
   robotOn(R,X,Y,T,D) % Expresa que un robot R está en X,Y en el tiempo T mirando hacia
   ↪  D.
   ```

   Esto es importante para que el visualizador funcione correctamente.
2. Asegurarte de que un robot solo pueda moverse hacia donde está mirando.
3. Implementar el giro de 90 grados para cambiar de dirección.

2.2. Actividad 2: Minimización de Energía

Ya implementada la direccionalidad, se busca hacer el proceso más eficiente. Ahora cada acción que realice un robot, ya sea moverse, cambiar de dirección o levantar una caja, tiene un costo de energía. La energía se gasta al ejecutar cada acción. Cuando se mantiene la ubicación no hay gasto energético. Es importante que tomes las siguientes consideraciones.

1. Se debe minimizar el número de acciones de cada robot para ahorrar energía.
2. Deben evitarse las acciones innecesarias, como moverse de un lado a otro sin propósito.
3. Se debe modelar el gasto de energía de cada robot en función de sus acciones.
4. Crear otro parámetro para el predicado `robotOn`, correspondiente a la energía de cada robot. La energía debe encontrarse en la última posición de robotOn:

   ```
   robotOn(R,X,Y,T,D,E) % Expresa que un robot R está en X,Y en el tiempo T mirando
   ↪  hacia D y con E energía restante.
   ```

   Esto es importante para que el visualizador funcione correctamente.

2.3. Visualizador

Para esta parte de la tarea, dispondrás de un visualizador temporal de la planificación que entregue tu programa. Ahora el predicado `robotOn` deberá tener dos parámetros extra al final que indique la direccionalidad y energía restante del robot. Es importante tener en cuenta que los test cases son incrementales, por lo que es necesario haber completado la Actividad 1 para que funcionen los tests de la Actividad 2.
Deberás mostrar los siguientes predicados en tu archivo de solución:

```
#show time/1.
#show boxOn/5.
#show obstacle/2.
#show rangeX/1.
#show rangeY/1.
#show goal/2.
#show robotOn/5.   % Cuando solo se haya implementado la direccionalidad.
#show robotOn/6.   % Cuando se hayan implementado la direccionalidad y energía.
```

Para hacer uso de esta funcionalidad, corre los tests 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*.

3. DCCudoku Tetris (2 pts.)

En esta parte, usaremos una variante del juego Sudoku que incorpora elementos del conocido juego Tetris. En lugar de los números convencionales, en esta variante se colocan piezas de Tetris en el tablero, cada una con números en su interior. Basado en el juego Flow Fit: Sudoku[1], llamamos a este juego *DCCudoku Tetris*.

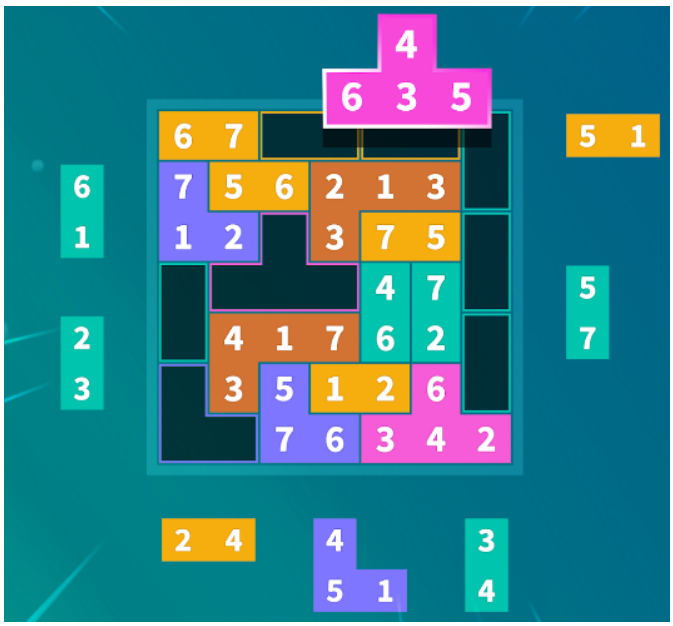


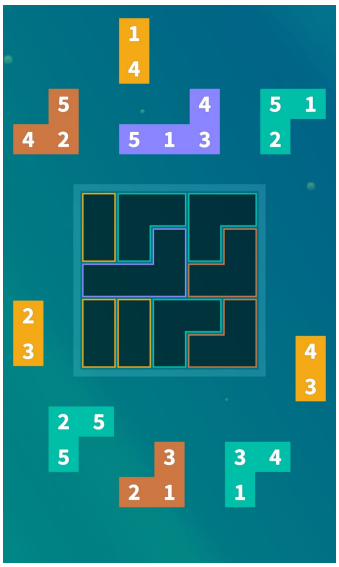
Figura 2: Juego DCCudoku

Para esta parte de la tarea, deberás adaptar un programa en lógica para resolver un DCCudoku Tetris. Las reglas del juego son simples:

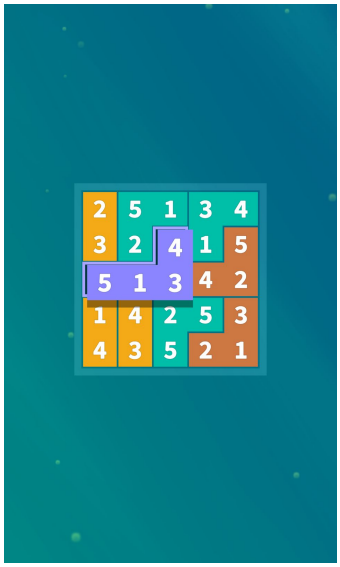
- Se juega sobre una grilla cuadrada de N x N.
- Existirán piezas de Tetris (que llamaremos Figuras) que se deberán colocar en la grilla respetando no superponer las figuras, y no salirse de la grilla
- Cada figura tiene números en su cuadrícula. A cada cuadrícula la llamaremos Bloque.
- Los números posibles que tienen los bloques irán de un rango de 1 a N.
- Basándose en el sudoku, en cada fila no puede existir un número repetido.
- De la misma manera, en cada columna no puede existir un número repetido.
- A diferencia del Sudoku original, **en esta variantes no existen "sub-grillas cuadradas"** donde se tenga que tener una vez cada número. Solamente interesa que no se repitan en filas y columnas.
- Un bloque no puede separarse de su figura, ni cambiar de posición relativa al resto de su figura (no se puede rotar ni cambiar de forma).

Siguiendo estas reglas, el objetivo del juego es posicionar todas las figuras de forma válida, sin dejar ningun espacio libre.

[1] https://play.google.com/store/apps/details?id=com.bigduckgames.flowfitsudoku



(a) Grilla al comienzo del juego



(b) Grilla al final del juego con la solución

Figura 3: Visualización del Sudoku Tetris original

Esta parte de la tarea será corregida de forma automática. Es por esto que es necesario dejar algunos elementos definidos de antemano para que todos lleguen a la misma solución.

IMPORTANTE: Utiliza los mismos predicados que serán sugeridos a continuación. De otra manera, tu solución puede que no sea leída por el programa que corregirá la tarea. Tu solución **no deberá** incluir ninguna regla que defina un puzzle específico. Es decir, no debe incluir hechos que describan figuras, bloques, posiciones, grilla.

3.1. Instancia

Una instancia del juego corresponde a un mapa del problema que tu programa deberá resolver. Los distintos mapas de juego se encontrarán dentro de la carpeta `tests/`, cada uno corresponderá a una grilla de $N \times N$ y se describirá de la siguiente forma:

```
grid(0..N-1, 0..N-1)
valores(1..N)
```

De esta manera `grid(X,Y)` es verdadero cuando `(X,Y)` es una posición de la grilla. Cada *ficha* tetris disponible para jugar está compuesta por *bloques*, cada uno de los cuales tiene asociado un valor numérico. Los $N \times N$ bloques existentes se declaran de la siguiente forma en el programa:

```
bloque(B, F, X_r, Y_r, V)
```

Donde `B` es el identificador del bloque, `F` es el identificador de la figura a la que pertenece el bloque, `X_r` y `Y_r` corresponden a la posición del bloque relativa a la figura, y finalmente `V` corresponde al valor numérico que tiene el bloque.

Por ejemplo, si la figura 4 tiene identificador 1 y está compuesta por los bloques [1, 2, 3, 4] y sería definida mediante:

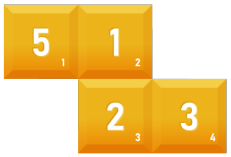


Figura 4: Bloque de ejemplo DCCudoku

```
% Figura 1
bloque(1, 1, 0, 0, 5).
bloque(2, 1, 0, 1, 1).
bloque(3, 1, 1, 1, 2).
bloque(4, 1, 1, 2, 3).
```

En resumen, se entregarán como "input" el tamaño `N` de la grilla, la grilla, los valores posibles y los bloques existentes. Dados estos hechos, tendrán que definir las reglas que permitan resolver el sudoku respetando las condiciones del juego ya mencionadas.

3.2. Resultado

Para formular el resultado de tu programa deberás utilizar el predicado

```
solucion(X, Y, V, B, F)
```

que deberá ser parte de un modelo si y solo si en la solución del juego el bloque `B` con valor `V` perteneciente a la figura `F` está en la posición `(X, Y)` de la grilla.

Puedes usar todos los predicados auxiliares que quieras, pero el programa que revisará tu archivo solamente tomará los predicados `solucion` para revisar que las piezas se colocaron correctamente.

3.3. Visualizador

Para facilitar el trabajo en la tarea, les entregaremos el código de un visualizador capaz de mostrar las soluciones que se obtienen por clingo. Para hacer uso del visualizador debes:

- Una vez tengas un programa capaz de solucionar el problema, digamos **solucion.lp**, debes colocarlo en la carpeta base **/DCCudoku**.
- Dentro de la carpeta **/DCCudoku**, ejecuta el comando

  ```
  clingo solucion.lp | python parser.py output.json
  ```

  De esta forma generarás un archivo llamado **output.json** dentro de la carpeta.
- Abre el archivo **index.html** y selecciona el archivo **output.json** en la página que se muestra. Finalmente este deberá ser seleccionado en el visualizador y mostrará la instancia de juego acorde a la solución encontrada.