



PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE

Facultad de Ingeniería

IIC2613 Inteligencia Artificial

2025-2

Tarea 2

Programación Lógica para el Razonamiento

Fecha de entrega: viernes 12 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 viernes 12 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 apoyarse de los [apuntes no oficiales del curso](#) para el desarrollo de la tarea, además de la bibliografía oficial del curso. En particular el uso de Clingo y modelación de problemas.

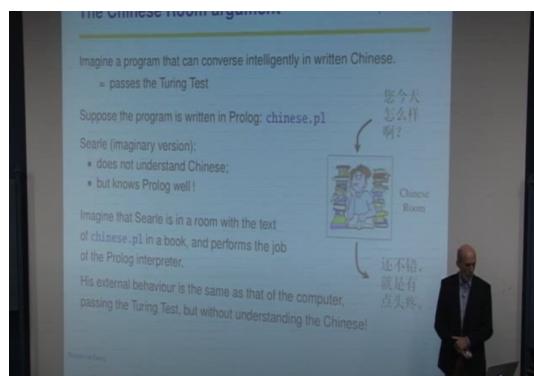
1. Reflexión y Teoría (1 punto)

Actualmente existen diversos podcasts disponibles en YouTube que discuten la teoría y aplicaciones de la IA. Dos de ellos son *Machine Learning Street Talk* y el podcast de Lex Fridman. En esta pregunta te ayudaremos a familiarizarte con este tipo de contenido en el contexto de nuestro curso.

1. **(0.5 puntos)** El tema de la conciencia en la Inteligencia Artificial ha ocasionado debate desde los inicios del área. Ve el extracto de la entrevista ([click acá](#)) del podcast *Machine Learning Street Talk* a François Chollet a principios de este año. Escucha entre 58:30 y 1:01:15 (aunque recomendamos seguir escuchando al menos 3 minutos más!). Escribe de la mejor forma que puedas cuál es la manera en que Chollet define al Sistema 2 como intrínsecamente ligado a la conciencia. ¿Es la definición de Chollet convincente para ti? Justifica. Ahora, toma una definición de conciencia entregada por la psicología, la filosofía o la espiritualidad (citando fuentes) y contrástala con la de Chollet. ¿Consideras que es posible que las máquinas adquieran ese tipo de conciencia? Escribe tu respuesta en no más de media página¹.



2. **(0.5 puntos)** Un problema filosófico de gran relevancia es si las máquinas podrían tener una “mente”, tal como la de los humanos. En respuesta a si las máquinas pueden o no pensar, Turing, al proponer su test, se conforma con que basta con que una máquina actúe como que piensa para darle el crédito de que piensa. Sin embargo, desde la filosofía ha habido fuerte oposición a esa postura. Ve ahora la presentación de Hector Levesque sobre *dos perspectivas del Test de Turing* ([click acá](#)) específicamente entre los minutos 14:57 y 27:55. Explica en qué consiste el argumento de la pieza china y el argumento sobre el sistema. Luego comenta cuál es tu punto de vista sobre estos argumentos. Escribe tu respuesta en no más de una página².



Qué entregar en esta parte

En un subdirectorio Parte1/ un archivo **respuesta.pdf** con tu respuesta.

¹Escrita a 11pt, fuente Arial/Times/Computer Modern. Recomendamos (no obligamos!) usar L^AT_EX para el informe.

²Ídem.

2. DCCursos (2 puntos)

Para poder obtener tu título de Ingeniero/a Civil, es necesario tomar una gran cantidad de ramos a lo largo de al menos 11 semestres. Cada vez se actualiza buscacursos te sientas, e intentas planificar tus ramos para tener un semestre equilibrado, que te permita avanzar en tu malla sin atrasarte. La tarea no es fácil, ya que existen cursos que son prerequisitos de otros, algunos que se dictan solo en semestres pares o impares, y quieres asegurarte de tomar entre 10 y 50 créditos (o entre uno y cinco ramos) cada semestre.

Reseterar malla		Exportar malla													
Semestre 1 50 créditos		Semestre 2 50 créditos		Semestre 3 50 créditos		Semestre 4 50 créditos		Semestre 5 50 créditos		Semestre 6 50 créditos		Semestre 7 50 créditos		Semestre 8 50 créditos	
QIM100A QUÍMICA GENERAL II 10 créditos	ICIS113 INTRODUCCIÓN A LA ECONOMÍA 10 créditos	OPTATIVO TERMODINÁMICA 10 créditos	OPTATIVO ELECTRICIDAD Y MAGNETISMO 10 créditos	ING2030 INVESTIGACIÓN, INNOVACIÓN Y EMPRENDIMIENTO 10 créditos	CURSO DE MAJOR 10 créditos	CURSO DE MAJOR 10 créditos	CAPSTONE CURSO DE MAJOR 10 créditos	MAT1610 CÁLCULO I 10 créditos	IC1103 INTRODUCCIÓN A LA PROGRAMACIÓN 10 créditos	FIS0152 LABORATORIO DE TERMODINÁMICA 0 créditos	FIS0153 LABORATORIO DE ELECTRICIDAD Y MAGNETISMO 0 créditos	OPTATIVO FUNDAMENTOS 10 créditos	CURSO DE MAJOR 10 créditos	CURSO DE MAJOR 10 créditos	CURSO DE MAJOR 10 créditos
ING1004 DESAFIOS DE LA INGENIERIA 10 créditos	MAT1620 CÁLCULO II 10 créditos	OPTATIVO DE EXPLORACION 10 créditos	OPTATIVO BIOLÓGICO 10 créditos	CURSO DE MAJOR 10 créditos	CURSO DE MINOR 10 créditos	CURSO DE MINOR 10 créditos	CURSO DE MINOR 10 créditos	MAT1203 ÁLGEBRA LINEAL 10 créditos	OPTATIVO ESTÁTICA Y DINÁMICA 10 créditos	MAT1640 ECUACIONES DIFERENCIALES 10 créditos	EYP1113 PROBABILIDADES Y ESTADÍSTICA 10 créditos	CURSO DE MAJOR 10 créditos	CURSO DE MINOR 10 créditos	CURSO DE MINOR 10 créditos	CURSO DE MINOR 10 créditos
FIL188 ÉTICA PARA INGENIEROS 10 créditos	FIS0151 LABORATORIO DE ESTÁTICA Y DINÁMICA 0 créditos	MAT1630 CÁLCULO III 10 créditos	CURSO DE MAJOR 10 créditos	OPTATIVO FORMACIÓN GENERAL 10 créditos	OPTATIVO FORMACIÓN GENERAL 10 créditos	OPTATIVO FORMACIÓN GENERAL 10 créditos	OPTATIVO FORMACIÓN GENERAL 10 créditos	LET0003 DESARROLLO DE HABILIDADES COMUNICATIVAS PARA INGENIEROS 10 créditos	OPTATIVO FORMACIÓN GENERAL 10 créditos	OPTATIVO FORMACIÓN GENERAL 10 créditos	OPTATIVO FORMACIÓN GENERAL 10 créditos	OPTATIVO FORMACIÓN GENERAL 10 créditos	OPTATIVO FORMACIÓN GENERAL 10 créditos	OPTATIVO FORMACIÓN GENERAL 10 créditos	OPTATIVO FORMACIÓN GENERAL 10 créditos

Figura 1: Ejemplo de antiguo planner.

Se te ocurre que el problema de planificar tus cursos puede resolverse de manera sencilla y eficiente aplicando los conceptos de *Answer Set Programming* (ASP) aprendidos en el curso de IA. Considerando los avances en programación de la IA, decides probar [Deepseek](#) para evaluar si es capaz de generar un código desde el cual comenzar a trabajar. En el prompt, le explicas a Deepseek que la información de los ramos de un semestre está descrita en un predicado `ramo_semestre`, de tal forma que si el programa contiene un hecho de la forma:

```
ramo_semestre(R, S).
```

se indica que el ramo R se toma el semestre S. A su vez, el predicado `requisito` describe la relación entre dos ramos, donde si se tiene el hecho:

```
requisito(R1, R2).
```

Se indica que el ramo R1 es prerequisito de R2.

A continuación, se presenta el código que te entregó Deepseek.

```
% ===== Hechos base (dominios) =====
%   ramo(1..N).
%   semestre(1..M).

% ===== Relaciones posibles =====
% requisito(R1, R2): el ramo R1 es prerequisito de R2.
```

```

% par(R): el ramo R se dicta solo en semestres pares.
% impar(R): el ramo R se dicta solo en semestres impares.

% ===== Reglas =====

{ramo_semestre(R, S) : ramo(R)} <= 5 :- semestre(S).

{ramo_semestre(R, S) : semestre(S)} = 1 :- ramo(R).

:- ramo_semestre(R2, S), not 1 { ramo_semestre(R1, N) : semestre(N), N < S,
→ semestre(N)}, R1 != R2, ramo(R1), ramo(R2), semestre(S), requisito(R1, R2).

:- ramo_semestre(R, S), S \ 2 = 1, par(R), ramo(R), semestre(S).
:- ramo_semestre(R, S), S \ 2 = 0, impar(R), ramo(R), semestre(S).

#maximize { R : ramo_semestre(R, _) }.

#show ramo_semestre/2.

```

Este código para la asignación de ramos lo puedes encontrar en el archivo `planner.lp` del subdirectorio `DCCursos/`: Contiene a su vez una serie de ramos y requisitos para la ejecución del código.

2.1. *Debugging del código generado por IA (0.5 puntos)*

Al analizar la solución generada por Deepseek, sospechas de la corrección de algunas reglas y del enfoque en general, dado que el programa te entrega configuraciones no factibles de ramos en algunos semestres.

Como aún deseas explorar el potencial de la inteligencia artificial, decides que la mejor forma de *debuggear* el código es consultando a [Chat GPT](#) sobre los posibles errores. Tras tu solicitud, la IA te proporciona la siguiente respuesta:

Resumen de Problemas y Soluciones

A continuación se presenta un resumen de los problemas identificados en el código ASP, junto con las respectivas soluciones propuestas.

1. Falta “R1 != R2” en la restricción de requisitos

Problema: En la restricción:

```
:‐ ramo_semestre(R2, S), not 1 { ramo_semestre(R1, N) : semestre(N), N < S, semestre(N) },  
ramo(R1), ramo(R2), semestre(S), requisito(R1, R2).
```

No se filtra que R1 y R2 sean distintos. Si hay una definición errónea (o un requisito reflexivo), puede generar resultados inconsistentes.

Solución: Agregar la condición:

```
:‐ ramo_semestre(R2, S), not 1 { ramo_semestre(R1, N) : semestre(N), N < S },  
requisito(R1, R2), R1 != R2.
```

o mejor aún, reemplazar por la versión más clara

```
:‐ requisito(R1, R2), ramo_semestre(R1, S1), ramo_semestre(R2, S2), S1 >= S2.
```

2. Restricción de máximo 5 ramos por semestre sin mínimo

Problema: La regla:

```
{ramo_semestre(R, S) : ramo(R)} <= 5 :- semestre(S).
```

solo limita el máximo (5), pero no impone un mínimo, lo que puede producir semestres vacíos o con muy pocos ramos, aunque se quieran distribuir equitativamente.

Solución: Definir un rango, por ejemplo entre 4 y 6 ramos:

```
4 { ramo_semestre(R, S) : ramo(R) } 6 :- semestre(S).
```

Esto asegura que cada semestre tiene al menos 4 y como máximo 6 ramos.

3. Paridad con expresión matemática complicada

Problema: Las restricciones

```
:‐ ramo_semestre(R, S), S \ 2 = 1, par(R), ramo(R), semestre(S).  
:‐ ramo_semestre(R, S), S \ 2 = 0, impar(R), ramo(R), semestre(S).
```

son confusas y poco claras. Además, $S \backslash 2$ en Clingo no es la mejor forma para expresar paridad, y puede fallar en versiones recientes.

Solución: Usar #mod para comprobar paridad:

```
:‐ ramo_semestre(R, S), S #mod 2 = 1, par(R). % Ramo par en semestre impar → prohibido  
:‐ ramo_semestre(R, S), S #mod 2 = 0, impar(R). % Ramo impar en semestre par → prohibido
```

4. “not 1 ... ” no es sintaxis válida en Clingo

Problema: El código original usa:

```
not 1 { ... }
```

Esto parece intentar decir “no hay al menos un elemento”, pero en Clingo no se escribe así. En Clingo se usan comparaciones en los agregados (#count, #sum) o condiciones como:

```
1 { ... } :- ...
```

Solución: Eliminar `not 1 { ... }` y usar comparación explícita con #count ^a:

```
:‐ ramo_semestre(R2, S2), requisito(R1, R2), #count { N : ramo_semestre(R1, N), N < S2 } = 0.
```

o la forma simplificada

```
:‐ requisito(R1, R2), ramo_semestre(R1, S1), ramo_semestre(R2, S2), S1 >= S2.
```

5. Uso redundante de “ramo(R1), ramo(R2), semestre(S)” en restricciones

Problema: En la restricción original hay literales redundantes:

```
ramo(R1), ramo(R2), semestre(S)
```

Estos ya están implícitos porque `ramo_semestre(R, S)` solo se cumple si R y S son ramos y semestres válidos. Esto hace que el código más largo y difícil de leer, sin aportar valor.

Solución: Eliminar redundancias:

```
:‐ ramo_semestre(R2, S), requisito(R1, R2), ramo_semestre(R1, N), N >= S.
```

^aPodría ser útil revisar cómo opera #count en el [libro](#), para comprender su uso correcto

Como primera actividad, revisa la respuesta proporcionada y analiza cada uno de los cinco problemas identificados. Para cada caso, determina si realmente representa un error en la formulación del problema. Si consideras que un problema debe ser corregido en el código, evalúa si la solución propuesta por Chat GPT es adecuada y justifica. En caso afirmativo, implementa la corrección correspondiente. En caso que consideres que hay una mejor solución, mencionalo y justifica. Tus respuestas deben ser concisas, con un máximo de cinco líneas por problema.

Para redactar tu respuesta, considera el contexto real de la universidad y la carrera de ingeniería al analizar los problemas sugeridos. Cualquier duda recuerda hacerla por las issues del curso o tomar cosas como supuestos y justificarlos debidamente.

Para esta sección debes entregar tu respuestas de desarrollo en un subdirectorio DCCursos/ en un archivo llamado **respuesta.pdf**, junto con un archivo **parte1.lp** con el código arreglado.

2.2. Documentación de código generado por IA (0.5 puntos)

Después de identificar y corregir todos los errores en el código, decides entregar tu planner a la universidad como una herramienta útil para las futuras generaciones. No obstante, consideras que podrían surgir preguntas sobre su funcionamiento.

Por esta razón, determinas que lo más adecuado es documentarlo. De este modo, no solo estarás preparada para aclarar cualquier interrogante, sino que también garantizarás que otros usuarios puedan comprender y utilizar el código de manera efectiva.

En esta actividad, deberás llevar a cabo lo siguiente:

1. Documenta el código que arreglaste en la etapa anterior, proporcionando para cada restricción una breve descripción (de no más de dos líneas) sobre su función y la razón por la que opera correctamente. La explicación de las reglas puede ser “literal” o precisa y en buen castellano. Apunta a lo segundo.
2. Solicita a ChatGPT que realice la misma tarea y adjunta su respuesta.
3. Compara tu documentación con la generada por ChatGPT y reflexiona sobre las diferencias y similitudes, señalando qué elementos de cada versión podrían mejorarse y cuál consideras más efectiva para futuros usuarios.

Para esta sección debes entregar en un subdirectorio DCCursos/ tus respuestas de desarrollo en el mismo archivo **respuesta.pdf** de la actividad anterior, junto con un archivo **parte2_gpt.lp** con el código comentado por ChatGPT y un archivo **parte2_alumno.lp** con el código comentado por ti.

2.3. Expansión y mejora de código generado por IA (0.5 puntos)

El código ahora funciona correctamente y cumple con el objetivo establecido. Sin embargo, los profesores han decidido añadir una nueva restricción para la toma de ramos, y te piden que la integres en el código.

Para seguir evaluando las capacidades de ChatGPT, le solicitas que incorpore la siguiente restricción:

Se debe procurar minimizar la cantidad de semestres para titularse, agregando un predicado max_semestre, que defina el último semestre con ramos.

En esta actividad, deberás llevar a cabo lo siguiente:

1. En el subdirectorio DCCursos/, en un archivo nuevo llamado `parte3_alumno.lp` copia el código anterior e implementa la restricción solicitada. Documenta tu solución proporcionando una breve descripción (de no más de dos líneas) sobre su función y la razón por la que opera correctamente. Verifica que funciona correctamente.
2. En el subdirectorio DCCursos/, en un archivo nuevo llamado `parte3_gpt.lp` copia el código anterior y solicita a alguna IA que realice la misma tarea y adjunta su respuesta. Verifica que funciona correctamente y explica por qué.
3. Compara tu solución con la generada por ChatGPT y reflexiona sobre las diferencias y similitudes, señalando qué elementos de cada versión podrían mejorarse.

Para esta sección debes entregar en un subdirectorio DCCursos/ tus respuestas de desarrollo en el mismo archivo `respuesta.pdf` de las actividades anteriores.

2.4. Reflexión y conclusiones (0.5 puntos)

Al desarrollar esta actividad, habrás notado que, a pesar de los grandes avances en inteligencia artificial, esta tecnología aún puede cometer errores. Por ello, es fundamental utilizarla con criterio y no depender ciegamente de sus resultados.

A continuación, plantea qué estrategia utilizarías para verificar la precisión y fiabilidad de los resultados y códigos generados por un programa de inteligencia artificial al momento de utilizarlo para generación, *debugging* y expansión de código. Define un protocolo simple para el correcto uso de inteligencia artificial en labores de programación.

Luego, reflexiona acerca del uso de estas tecnologías en el ámbito de tu educación, y de las consecuencias de utilizarla sin seguir la estrategia planteada anteriormente.

Para esta sección debes entregar en un subdirectorio DCCursos/ tus respuestas de desarrollo en el archivo `respuesta.pdf`.

Qué entregar en esta parte

En un subdirectorio DCCursos/:

- Un archivo `respuesta.pdf` con tus respuestas ordenadas.
- Para la parte 1 crea un archivo `parte1.lp` con su respectiva respuesta
- Para la parte 2 crea los archivos `parte2_alumno.lp` y `parte2_gpt.lp` especificados anteriormente.
- Para la parte 3 crea los archivos `parte3_alumno.lp` y `parte3_gpt.lp` descritos previamente.

3. DCConstruction Blocks (3 puntos)

3.1. Introducción: Un correo inesperado



Figura 2: Un correo inesperado te llega en la clase de IA.

Mientras estabas en la clase de Inteligencia Artificial, revisando distraídamente tu bandeja de entrada, un correo inesperado apareció con el asunto “*Herencia pendiente*”. Al abrirlo, descubriste que un misterioso familiar te había dejado como legado una fábrica llamada DCConstruction Blocks, dedicada a producir piezas encajables muy similares al LEGO que usabas de niño.

El mensaje era claro: la fábrica estaba en problemas. El proceso de crear instrucciones para armar estructuras seguía haciendo a mano, proceso que era lento y propenso a errores. Sin una solución, la empresa no sobreviviría.

Justo en ese instante, el profesor Jorge Baier comenzó a hablar de cómo ASP se podría usar para resolver aquellos problemas asociados al “sistema 2” del pensamiento humano. Entonces tu mente se iluminó: podías transformar ese proceso lento y costoso en un sistema optimizado con IA. Generar el conjunto de piezas necesarias, con sus colores y formas, ya no sería solo un trabajo humano, sino que podríamos combinar creatividad humana con automatización implementada por IA.

Así, entre el murmullo de la sala y la extrañeza del correo, comenzaba tu aventura: heredar una fábrica perdida en el tiempo y salvarla con las herramientas de la programación lógica.

3.2. DCConstruction Blocks: Un mundo de diversión

El objetivo DCConstruction Blocks es armar estructuras con bloques encajables de diferentes formas y colores dentro de una plataforma cuadrada. Cada caja de DCConstruction Blocks viene con un libro de instrucciones que indica las piezas que van en cada capa de altura, mostrando su forma y color; así como una vista superior de la estructura final deseada. Algo muy importante para estos *sets* es que todas las piezas de la figura final deben estar conectadas, de forma que uno pueda manipularlas sin que se desarmen. La lista de las piezas disponibles se encuentra a continuación:

1. Pieza **lineal de 2×1**: Puede orientarse tanto **vertical** como **horizontalmente** en la plataforma.
2. Pieza **lineal de 3×1**: Puede orientarse tanto **vertical** como **horizontalmente** en la plataforma.
3. Pieza **lineal de 4×1**: Puede orientarse tanto **vertical** como **horizontalmente** en la plataforma.

4. Pieza **cuadrado de 2×2** : Posee **orientación única**.
5. Pieza en forma de “L” de 2×3 : Posee **8 orientaciones** posibles.

Además, los colores posibles que puede tener cada pieza son 4: **amarillo, azul, verde o rojo**.

Por lo tanto, lo que queremos resolver es: dado una objetivo final en 2D, correspondiente a una vista superior de las piezas, encontrar una configuración de piezas, ubicadas posiblemente a distinta altura y conectadas entre ellas cuya vista superior coincida con la dada. A continuación un ejemplo de esto:

	0	1	2	3	4
0		A	A	A	
1	R	V	R		
2		A	A	A	
3					
4					

Figura 3: Ejemplo de objetivo. Vista desde $Z \geq 2$.

	0	1	2	3	4
0					
1	L	L	L		
2			L		
3					
4					

	0	1	2	3	4
0		L	L	L	
1	L				
2	D	D	D		
3					
4					

	0	1	2	3	4
0		I	I	I	
1					
2					
3					
4					

(a) Pieza ubicada en $Z = 0$ (b) Piezas ubicadas en $Z = 1$ (c) Pieza ubicadas en $Z = 2$

Figura 4: Ejemplo de solución para objetivo.

3.3. ¿Qué te entregaremos?

Para poder completar los 3 desafíos y así conseguir automatizar el proceso de creación de instrucciones, te entregaremos lo siguiente:

- **Código Base:** Un archivo `dcconstruction.lp` con reglas básicas para el correcto funcionamiento del código, que incluye la definición de las dimensiones del tablero y los colores de las piezas. A partir de este código deberás implementar tu solución para los diferentes desafíos.
- **Estructura objetivo a la que tienes que llegar:** Te entregaremos la información de la estructura final que deberás armar mediante el predicado `objetivo(X, Y, C)`, que espera que el **color superior** de la casilla (X, Y) sea C . Por ejemplo, el código

```

objetivo(1,2,amarillo).
objetivo(0,1,azul).
objetivo(1,1,azul).
objetivo(2,1,azul).

```

espera que entregues la estructura de la Figura 5.

	0	1	2	3	4
0					
1	A	A	A		
2		A			
3					
4					

Figura 5: Ejemplo de vista superior de estructura coloreada en tablero de 5x5.

- **Visualizador de resultados de Clingo:** Contarás con un *script* de visualización para poder ver cómo quedó tu estructura. Para saber más, revisa la Sección 3.11

IMPORTANTE: El proceso de revisión será mediante este *script* de visualización que esperará que le pases los predicados **bloque**, **colocacion** y **color_superior**, los cuales **DEBEN** estar presentes en tus programas.

3.4. Predicados que debes utilizar

Para poder completar los 3 desafíos y así conseguir automatizar el proceso de creación de instrucciones, te pedimos que uses los siguientes predicados:

- **Definición de piezas:** Se detalla en la sección 3.5
- **Definición de colocaciones de piezas:** Para definir que una determinada pieza se ubicó en una casilla específica, deberás usar el predicado **colocacion(X, Y, Z, T, C)** donde (X, Y, Z) es la coordenada (tridimensional) donde se va a colocar la pieza, T es el nombre de la pieza y C es el color de la pieza.
- **Definición del color superior:** Para los casos donde haya una pieza sobre otra (Desafío 3) deberás determinar cuál es el color visible en cada casilla de tu estructura si ésta se mirase desde arriba. Para ello, deberás contar con el predicado **color_superior(X, Y, C)** que indica que en la casilla (X, Y) el color de la pieza superior (con mayor valor en la coordenada Z) es C.
- **Definición del límite de piezas disponibles:** En cada test se indicará la cantidad máxima de esas piezas que se pueden usar mediante el predicado **cupo(T, num)**, donde T es el nombre de la pieza y num es un entero que indica la cantidad máxima de piezas con ese nombre que se pueden emplear. Deberás crear nuevas restricciones a partir de este predicado.

3.5. Desafío 0: Definiendo piezas (0.25 puntos)

Para definir tus piezas de construcción deberás crear 2 predicados: **forma** y **bloque**.

- **forma(T):** Define el nombre T (y existencia) de una pieza. Por ejemplo, **forma(linea2h)** definiría la existencia de la pieza lineal de 2x1 con orientación horizontal y con nombre linea2h.
- **bloque(T, DX, DY):** Define el desplazamiento en los ejes X e Y en los cuales una pieza T estará ocupando un espacio. Por ejemplo, si se especifica:

```

forma(linea2h).
bloque(linea2h, 0, 0).
bloque(linea2h, 1, 0).

```

entonces, al posicionar una pieza lineal de 2×1 en la coordenadas (X, Y) , esta pieza ocupará efectivamente las casillas $(X + 0, Y + 0)$ y $(X + 1, Y + 0)$.

En esta sección deberás definir todos las piezas detalladas en la Sección 3.2. Para el correcto funcionamiento debes nombrar de la siguiente forma a tus piezas:

1. Cuadrado de 2×2 :

Este debe llamarse cuadrado2x2. Recuerda que esta pieza no tiene orientación.

2. Piezas lineales:

Las piezas lineales pueden estar orientadas tanto vertical como horizontalmente y pueden tener un tamaño entre 2 a 4 celdas. Estas deben llamarse lineal<tamaño><orientación>, donde tamaño $\in \{2, 3, 4\}$ y orientación $\in \{v, h\}$. Por ejemplo, la pieza vertical de 3×1 debe llamarse lineal3v.

3. Piezas “L” de 2×3 :

Las piezas “L” tienen un tamaño único de 2×3 . Estas pueden tener 8 orientaciones posibles dependiendo de rotación y reflexión. Estas deben llamarse 12x3<orientación> donde orientación $\in \{\text{sur}<1,2>, \text{norte}<1,2>, \text{este}<1,2>, \text{oeste}<1,2>\}$. A continuación una imagen de las piezas:

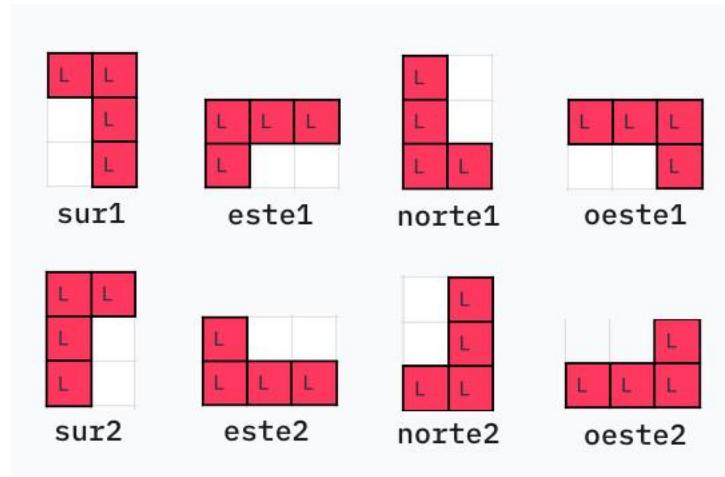


Figura 6: Orientaciones posibles de pieza L 2×3 .

Entregable: Estos predicados serán usados en las siguientes actividades, por lo que deben ser incluidos en todas ellas.

3.6. Desafío 1: Dibujando contornos (0.75 puntos)

Como siguiente desafío, vamos a olvidarnos de las complejidades de trabajar en 3 dimensiones y con varios colores. Deberás **asumir que solo existe 1 coordenada Z válida y sólo cuentas con el color amarillo** para generar las estructuras pedidas.

La idea en este desafío es que seas capaz de dibujar sobre contornos que te sean dados (con el predicado objetivo, mencionado en la Sección 3.3), con las piezas que fueron mencionadas en la Sección 3.2 y que debiste definir en el Desafío 0. Para ello, te recomendamos tener en cuenta para tu programa en Clingo lo siguiente:

1. Respeta las restricciones en la cantidad de piezas.
2. Evita que las piezas se salgan del tablero.
3. Asegúrate no haya dos piezas en la misma coordenada.

Entregable: Para este desafío deberás entregar en el subdirectorio DCConstruction/ un archivo de Clingo llamado desafio1.lp con tu respuesta.

3.7. Desafío 2: Coloreando y optimizando (1 punto)

Como segundo desafío, ahora deberás agregarle color a las estructuras (ya no solo amarillo). Para ello, **se te pedirá que una casilla tenga alguno de los 4 colores mencionados al final de la Sección 3.2:** amarillo, azul, rojo o verde.

Además, como las estructuras se están complejizando (y no queremos gastar material innecesario en piezas extra), se vuelve necesario que **optimices tus soluciones** de colocación de piezas. Para ello, tendrás que **implementar los siguientes dos objetivos de optimización:**

1. Minimizar la cantidad total de piezas colocadas en tu estructura.
2. Maximizar la cantidad de bloques que ocupa cada pieza colocada.

Para ello, te sugerimos investigar cómo usar las directivas de optimización #maximize y #minimize disponibles en Clingo para que puedas usarlas en tu código.

Finalmente, deberás **garantizar que aún se cumplan las restricciones del Desafío 1**, por lo cual te recomendamos usar el código del desafío anterior como base.

Entregable: Para este desafío deberás entregar en el subdirectorio DCConstruction/ un archivo de Clingo llamado desafio2.lp con tu respuesta.

3.8. Desafío 3: Aumentando las dimensiones (1 punto)

El último desafío para que termines la automatización de la elaboración de instrucciones es permitir que las piezas se conecten (se encajen) entre sí. Para ello, ahora deberás trabajar con la coordenada Z de tus piezas, de modo que **todas las piezas conformen una estructura conexa**. De esta manera, si se quisiese levantar la estructura, sin importar desde cuál pieza se tome, ésta no se desarmará³.

Una cualidad interesante de este problema es que **se puede modelar la conexión entre piezas como un grafo**, donde cada pieza es un nodo y, si hay una arista entre nodos, entonces entenderemos que esto representa que las piezas asociadas están conectadas (visualmente, una estaría sobre la otra).

De esta manera, tu modelo final **deberá generar un grafo conexo⁴ entre las piezas de tu estructura**, garantizando que **aún se cumplan las condiciones que ya te pedíamos en los desafíos anteriores**. A saber, aún deberás

1. Respetar las restricciones en la cantidad de piezas.
2. Evitar que las piezas se salgan del tablero.
3. Asegurarte que no haya dos piezas en la misma coordenada (observa que esta restricción ahora **debe extenderse a 3 coordenadas**).
4. Respetar los colores que se te pide en el predicado objetivo (y ahora también deberás modificar tu predicado **color_superior** de modo que obtenga el color visible desde arriba de cada casilla correctamente).

³Ignorando el peso propio de la estructura, claro.

⁴Puedes *googlear* más sobre este tema para apoyarte en el desarrollo de tu tarea.

5. Optimizar tu solución final para que contenga la menor cantidad de piezas posibles y que estas sean las de mayor tamaño.

Por lo anterior, te recomendamos que tomes como base tu código del Desafío 2 y lo modifiques para que cumpla las nuevas restricciones.

Entregable: Para este desafío deberás entregar en el subdirectorio `DCConstruction/` un archivo de Clingo llamado `desafio3.lp` con tu respuesta.

3.9. (Bonus) Desafío 4: Extensión a packs 3 en 1 (0.3 puntos)

Luego de implementar el programa de *DCConstruction Blocks*, recuerdas que cuando eras niño solías jugar con unos bloques muy similares. En particular te acuerdas de unos sets llamados 3 en 1, que al reordenar las piezas contenidas, permitían formar hasta 3 figuras diferentes (puedes ver un ejemplo haciendo click aquí⁵). Debido a esto, decides modificar tu código para poder replicar estos sets, de manera que permita formar 3 figuras diferentes con las mismas piezas.

Para esta sección, deberás **únicamente describir** cómo modificarías tu programa `desafio3.lp` para formar sets 3 en 1. Describe como debe ser el input y output de este programa, así como todas las reglas que debes agregar/modificar para lograrlo⁶.

Entregable: Para este desafío deberás entregar en el subdirectorio `DCConstruction/` un archivo de Clingo llamado `bonus.pdf` con tu respuesta.

3.10. Tests

Para que puedas ir revisando si tu tarea está yendo por buen camino, te entregaremos un conjunto de *tests* ya preparados. Estos están organizados en el directorio `tests/`, el cual contiene tres subcarpetas, una para cada desafío:

- `tests/D1/` → Contiene los *tests* para el Desafío 1.
- `tests/D2/` → Contiene los *tests* para el Desafío 2.
- `tests/D3/` → Contiene los *tests* para el Desafío 3.

En total, te entregamos **12 tests (4 por desafío)** que te permitirán verificar el progreso de tu implementación. Estos *tests* están diseñados para aumentar progresivamente en dificultad (el test 1 es el más fácil y el test 4 el más complejo), de manera que vayas construyendo tu solución paso a paso.

IMPORTANTE: Si tu tarea pasa todos los *tests* no implica que tendrás todo el puntaje. Puede suceder que no pases alguno de los *tests* privados que tendremos para corregir. Por ello, te invitamos a crear tus propios *tests* para revisar aún mejor tu código.

Además, considera que en algunos casos tu programa puede no encontrar una solución óptima en el tiempo límite dado. En tales situaciones, te recomendamos **introducir podas o restricciones adicionales que reduzcan el espacio de búsqueda**, siempre y cuando:

1. No alteren las reglas centrales ya descritas en el enunciado.
2. No eliminan posibles soluciones válidas ni modifiquen el resultado correcto.

Esto forma parte del proceso de modelado: encontrar maneras de reducir el tiempo de cómputo sin comprometer la corrección de la solución.

⁵<https://www.tiendalego.cl/collections/creator-3in1/products/robot-espacial-31164>

⁶Escrita a 11pt, fuente Arial/Times/Computer Modern. Recomendamos (no obligamos!) usar L^AT_EX.

3.11. Funcionamiento del visualizador

Te entregaremos un visualizador para que veas las soluciones de tu programa. Para esto deberás ejecutar el siguiente comando:

```
python3 visualizer.py [archivo.lp] [test.lp] [salida.html]
```

Donde `archivo.lp` corresponde al archivo de Clingo que quieras probar con su respectivo test contenido en el archivo `test.lp`. Por último, `salida.html` corresponde a un archivo HTML que será generado con la visualización correspondiente, por lo que puedes nombrarlo como prefieras.

Luego debes abrir el archivo HTML generado en algún navegador, donde podrás interactuar con la visualización. A continuación un ejemplo de se ve este:



Figura 7: Ejemplo del visualizador.

Es muy importante que se incluyan los siguientes *statements* `#show` inicialmente entregados en el archivo `dcconstruction.lp`:

```
% Para que el visualizador funcione debes dejar los siguientes show:  
#show ancho/1.  
#show alto/1.  
#show colocacion/5.  
#show bloque/3.  
#show color_superior/3.
```

Por último, para comenzar a resolver el problema puede ser útil querer manipular solamente el archivo de Clingo sin el visualizador y ver la salida en la consola. Para esto se puede utilizar el siguiente comando:

```
clingo tests/<dificultad>/test<X>.lp desafio<1,2,3>.lp
```

3.12. Entrega

- Tu programa **no deberá** incluir ninguna regla que defina un problema específico. Es decir, no debe incluir hechos que describan órdenes, posiciones, etc.
- En un subdirectorio `DCConstruction/`: Un archivo llamado `desafio1.lp` con el programa de la sección 1.
- En un subdirectorio `DCConstruction/`: Un archivo llamado `desafio2.lp` con el programa de la sección 2.

- En un subdirectorio DCConstruction/: Un archivo llamado desafio3.lp con el programa de la sección 3.