

PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
Escuela de Ingeniería
IIC2613 – Inteligencia Artificial
2025-1

Interrogación 1

Miércoles 23 de abril
Duración: 1:45 hrs.

Aspectos generales

Formato y plazo de entrega

El formato de entrega será en papel, subiendo sus respuestas para las preguntas de desarrollo a Canvas en forma de fotografías (se recomienda utilizar una aplicación para escanear con el teléfono), además, deberás reportar tu asistencia a la interrogación antes de abandonar la sala al cuerpo docente.

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 para este control debe ser **totalmente individual**. Tus dudas se deben ser consultadas exclusivamente al cuerpo docente levantando tu mano para ser atendido/a. Cualquier falta a la integridad académica será sancionada con la reprobación del curso y los antecedentes serán entregados a la Dirección de Pregrado.

0.1. Esquema de Evaluación

Para cada pregunta habrán 2 opciones de puntaje. La primera es puntaje completo, el cual se otorgará cuando la solución este completamente correcta, La segunda opción es puntaje parcial para una solución que tenga un error menor. Solo recibirán puntajes las respuestas que estén en estas categorías.

1. Para representar el funcionamiento de un sistema de regadío de una huerta controlada por IA, un programador ha escrito las siguientes reglas.

```
regador(r1; r2; r3).
planta(oregano; tomate; apio; espinaca).
apunta(r1,oregano).
apunta(r1,apio).
apunta(r2,tomate).
apunta(r3,espinaca).
regada(P) :- regador(R), encendido(R), apunta(R,P).
```

   a) (1 punto. Puntaje parcial (pp): 0.7 pts.) Escribe una regla que exprese el conocimiento *"cuando llueve todas las plantas se riegan"*.

   Respuesta: `regada(P) :- planta(P), llueve.`

   b) (1 punto. PP: 0.7 pts.) Escribe el predicado `falta_alguna`, que se deduce del programa cuando al menos una planta no está regada.

   Respuesta: `falta_alguna :- planta(P), not regada(P).` Debe estar la negación.

   c) (1 punto. PP: 0.7 pts.) Escribe el predicado `todas_regadas` que se cumple cuando todas las plantas están regadas. (Ayuda: este predicado se puede escribir en términos de `falta_una`.

   Respuesta: `todas_regadas :- not falta_alguna.` Debe estar la negación.

   d) (2 puntos. PP: 1.5 pts.) Describe qué reglas es necesario agregar al programa para que clingo nos entregue cuáles regadores encender para que todas las plantas estén regadas. Si requieres hacer un supuesto, déjalo por escrito.

   Respuesta: Supondremos que le permitiremos a clingo encender cualquier número de regadores. Entonces agregamos las siguientes 2 reglas:

```
{encendido(R) : regador(R)}.
:- not todas_regadas.
```

   Un error menor en este caso consiste en errores de sintaxis. Deben estar al menos ambas lineas.

2. Las variables en programación ASP permiten un alto poder de expresividad. Sin embargo, al invocar al solver, éste las elimina antes de comenzar a resolver. Específicamente, esto se traduce en que el programa:

```
p(a).
p(b).
q(X) :- p(X).
```

   Es transformado en el siguiente programa antes de resolver:

```
p(a).
p(b).
q(a) :- p(a).
q(b) :- p(b).
```

   Diremos que esto significa que la regla `q(X) :- p(X)` *se expande* en 2 reglas.

   Ahora vamos al contexto de los ejemplos de clases y tarea, en donde debimos escribir restricciones para que los robots no se pudieran 'atravesar'. En clases hablamos de que habían dos forma de escribir esto. En la primera forma, prohibimos un intercambio de robots de la siguiente manera:

```
:- robot(R1,X,Y,T), robot(R2,Xp,Yp,T), robot(R2,X,Y,T+1), robot(R1,Xp,Yp,T+1), R1!=R2.
```

   En la segunda forma, lo escribimos así:

```
:- robot(R1,X,Y,T), robot(R2,X+1,Y,T), robot(R2,X,Y,T+1), robot(R1,X+1,Y,T+1).
:- robot(R1,X,Y,T), robot(R2,X,Y+1,T), robot(R2,X,Y,T+1), robot(R1,X,Y+1,T+1).
```

a) (2 puntos. PP: 1.5 pts.) Si la grilla tiene $N \times N$ celdas, calcula en cuántas reglas se expande la primera y en cuántas se expande la segunda forma, en términos de N. En base a eso, emite una opinión justificada respecto de cuál es la forma más eficiente de representar esta restricción.

Respuesta: Como supuesto, consideramos que el número de robots y el makespan (es decir, el rango de la variable `T` es constante, por lo que no los incluimos en el análisis). La primera forma se expande en el número pares de celdas distintas que hay en el mapa, es decir $N^2(N^2-1)$. En la segunda forma, se expande en una restriccion por cada par de celdas horizontales adyacentes y una por cada par de celdas verticales adyacentes, lo que da un total de $2N(N-1)$. De esto se concluye que la forma más eficiente de representarlo es la segunda porque se expande en menos restricciones para representar el mismo conocimiento. Si no se consideró el número de robots o el bouns constante, la respuesta igualmente puede estar correcta si es que eso se inculye en el análisis. Los errores menores incluyen errores al contar o errores en la expresión como $N^2 - 2$, pero debe estar el análisis de cuadrático por cuadrático y lineal por lineal

b) (2 puntos. PP: 1.4 pts.) Explica por qué en la primera forma es necesario utilizar `R1!=R2`, y por qué en la segunda no lo es.

Respuesta: La razón es que en la primera, de no hacerlo, se puede instanciar `R1=R2`, al mismo tiempo que `X=Xp` y `Y=Yp`. Con esto, la restricción dejaría insatisfactible al programa si es que se requiere ejecutar un *wait*.

En la segunda no es necesario hacer la distinción porque `R1` y `R2` no se pueden instanciar con el mismo robot, esto es porque en tiempo `T`, la restricción pide que estén en lugares distintos; específicamente en `(X,Y)` y `(X+1,Y)`.

Deben estar ambas ideas para obtener el puntaje parcial. Si la explicación no es clara o detallada se otorga puntaje parcial

3. a) Dado un programa Π sin variables, ni negación, ni restricciones de cardinalidad, decimos que M es un modelo de Π si M es un conjunto minimal respecto de la relación $\subseteq$ que es tal que si $H \leftarrow B \in \Pi$ y $B \subseteq M$, entonces se cumple que $H \cap M \neq \{\}$.

Supón que $\Pi = \{\{p, q\} \leftarrow \{r\}, \{r\} \leftarrow \{\}\}$.

i. (3 puntos. PP: 2.3 pts.) Demuestra que $\{p, r\}$ y que $\{q, r\}$ son modelos de Π.
Respuesta: La primera observación es que $r \in M$. Esto es porque $\{r\} \leftarrow \{\} \in \Pi$, lo que implica que $\{r\} \cap M \neq \{\}$ (equivalente a $r \in M$).
Dado que $r \in M$ y que $\{p, q\} \leftarrow \{r\} \in \Pi$, la definición nos pide que $\{p, q\} \cap M \neq \{\}$, lo que se cumple si M un elemento entre $\{p, q\}$, por lo que $M = \{p, r\}$ o $M = \{q, r\}$.
Deben estar ambas observaciones para obtener el puntaje parcial. Se debe usar la defincón de programa. Si la explicación no es clara o detallada, o hay errores en el desarrollo se otorga puntaje parcial

ii. (1 punto. PP: 0.7 pts.) ¿Es $\{p, q, r\}$ también un modelo de Π? Justifica.
Respuesta: No es un modelo por minimalidad, dado que $\{p, r\} \subset \{p, q, r\}$ ya es modelo.
Se debe incluir minimalidad con la explicación de los subconjuntos para obtener puntaje parcial.

b) (3 puntos. PP: 2.3 pts.) Demuestra que si M es un modelo de un programa que contiene la línea:

```
:- not p.
```

entonces M contiene a `p`, usando la definición de modelo de un programa con negación. (Ayuda: puedes realizar la demostración por contradicción, suponiendo que el programa completo tiene un modelo M que es tal que $p \notin M$.)

Respuesta: Sea Π el programa que contiene a la regla en cuestión y sea M es uno de sus modelos. Dado que Π es un programa con negación, debería darse que M es modelo de Π^M.

Demostraremos lo que se pide usando un argumento de contradicción. Supongamos que $p \notin M$. Al computar la reducción Π^M, dado que $\{\} \leftarrow \{not\ p\} \in \Pi$, obtenemos que Π^M contiene a la regla $\{\} \leftarrow \{\}$. Por la definición de modelo de un programa con negación, M es un modelo de Π^M. Como $\{\} \leftarrow \{\} \in \Pi^M$ y $\{\} \subseteq M$, entonces se debe dar que $M \cap \{\} \neq \{\}$, lo que es una contradicción. Esto implica que Π^M no tiene ningún modelo lo que es una contradicción con el hecho de que M es modelo de Π.

Se debe utilizar la definición de reducción para obtener puntaje parcial. Si la explicación no es clara o detallada, o hay errores en el desarrollo se otorga puntaje parcial

4. Para cada una de las siguientes afirmaciones, di si son verdaderas o falsas, **justificando** tu respuesta a partir de un **contraejemplo** si la afirmación es falsa o con una **demostración clara** si la afirmación es verdadera.

   a) (2 puntos. PP: 1.5 pts.) El algoritmo BFS (de búsqueda en amplitud), siempre encuentra una solución óptima, independientemente de si los costos asociados a acciones diferentes son o no iguales.

   Respuesta: **Falso**. Como contraejemplo funciona cualquier árbol de búsqueda en donde un estado objetivo se encuentra a profundidad p y otro (o el mismo estado objetivo) se encuentra a profundidad $q > p$, pero donde el camino que llega al estado con q aristas es de costo menor al camino que llega al estado con p aristas. BFS siempre encuentra un estado objetivo que es alcanzable por un camino cuyo número de aristas es mínimo, lo que implica que encuentra una solución subóptima en costo.
   Para obtener puntaje parcial debe existir una justificación

   b) (2 puntos. PP: 1.5 pts.) Si h es una función heurística perfecta, es decir, $h(s)$ es igual al costo de un camino óptimo entre s y un estado objetivo, entonces el algoritmo *Greedy Best-First Search* siempre encuentra una solución óptima.

   Respuesta: Se puede argumentar que GBFS expande *solo* los estados que conducen a la solución óptima en orden hasta llegar al objetivo. Sin pérdida de generalidad, supongamos que tenemos un solo camino de costo óptimo $s_1 \dots s_n$. Al expandir s_i sabemos que $h(s_{i+1})$ es el menor valor heurístico de entre todos los hermanos de s_i (lo que se da gracias a que h es perfecta), esto significa que en la siguiente iteración s_{i+1} será expandido (y que su padre será marcado como s_i). Finalmente s_n será expandido en la iteración n y el camino óptimo será encontrado.
   Para obtener puntaje parcial debe existir una justificación que haga referencia al algoritmo GBFS

   c) (2 puntos. PP: 1.5 pts.) Sea h es una función heurística. El algoritmo *Greedy Best-First Search* (GBFS), ejecutado con wh ($w > 1$), expande el mismo número de nodos y no ejecuta más rápido que GBFS ejecutado con h.

   Respuesta: **Verdadero**. Dado que GBFS utiliza una heurística solamente como prioridad para la *Open*, el ranking generado por h es el mismo que con wh por lo que ambas ejecuciones funcionan exactamente igual.
   Para obtener puntaje parcial debe existir una justificación que haga referencia a que cambiar el peso asociado a la heurística no cambia el orden