PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
Escuela de Ingeniería
IIC2613 – Inteligencia Artificial
2024-2

Control Largo 1

Viernes 27 de septiembre
Duración: 1:30 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 al control antes de abandonar la sala al cuerpo docente, verificando que lo marquen dentro de la lista de asistentes.

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**. 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 levantando tu mano para ser atendido. 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.

Pregunta 1 (4 puntos)

(a) **(2 puntos)** Supón que Π es un programa en lógica que tiene n modelos. ¿Es correcto que al agregar la siguiente línea a Π:

```
1 {p ; q} 1.
```

el programa resultante tiene $2n$ modelos? Justifica tu respuesta.

Respuesta: No es correcto. Por ejemplo, cuando Π es

```
p.
```

el programa tiene un solo modelo.

(b) **(2 puntos)** Supón que Π es un programa básico (sin negación ni restricciones de cardinalidad, pero posiblemente con reglas con disyunciones en las cabezas) con m reglas. Sea M un modelo de Π. ¿Cuántos modelos de Π podrían existir que son subconjunto propio[1] de M? Justifica en base a la definición de modelo.

Respuesta: La definición de modelo expresa que todo modelo es minimal. Esto implica que no puede haber ningún modelo que es subconjunto propio de M. La repuesta, entonces, es 0.

Pregunta 2 (6 puntos)

(a) Usando la definición de modelo, demuestra que el siguiente programa no tiene modelos.

```
p.
:- p.
```

Respuesta: Supongamos que existe un modelo M para el programa. Por la primera regla, se debe cumplir que $p \in M$. Por la segunda regla, dado que $\{p\} \subseteq M$, se debe cumplir que $M \cap \emptyset \neq \emptyset$ lo que contradice el hecho que $p \in M$. Concluimos que no existe un modelo para Π.

(b) **(2 puntos)** ¿Cuántos modelos tiene el siguiente programa?

$$
\begin{aligned}
p_1 &:\text{-}\ not\ p_2.\\
p_2 &:\text{-}\ not\ p_1.\\
&\\
p_2 &:\text{-}\ not\ p_3.\\
p_3 &:\text{-}\ not\ p_2.\\
&\ldots\\
p_{n-1} &:\text{-}\ not\ p_n.\\
p_n &:\text{-}\ not\ p_{n-1}
\end{aligned}
$$

Justifica tu respuesta utilizando la definición de modelo.

Respuesta: Primero observamos que por la forma que está construido el programa, no es posible que tanto p_i como p_{i+1} pertenezcan simultáneamente a un modelo. Tampoco es posible que ambos, simultáneamente, no pertenezcan. Esto solo permite dos modelos $M_1 = \{p_i \mid i \text{ es par}\}$ y $M_2 = \{p_i \mid i \text{ es impar}\}$. Finalmente es directo observar que Π^{M_1} tiene como modelo a M_1 y que Π^{M_2} tiene como modelo a M_2.

[1] A es subconjunto propio de B cuando A es subconjunto de B pero no es igual a B.

(c) **(2 puntos)** Sea Π el siguiente programa

```
m :- not n.
p :- not q, not r, s.
t :- not u.
v :- x, not y.
```

Dado $X = \{\mathtt{n}, \mathtt{y}\}$, describe el contenido de Π^X, o, dicho de otra forma, escribe las reglas que tiene la reducción (o simplificación) de Π con respecto a X.

Respuesta:

```
p :- s.
t.
```

Pregunta 3 (8 puntos)

El objetivo de esta pregunta es modelar el razonamiento de un agente que se puede mover entre ubicaciones, y recoger y dejar dos objetos: una taza o una pelota.

Las ubicaciones se representan con un predicado `ubicacion`. Por ejemplo:

```
ubicacion(oficina).
ubicacion(cocina).
ubicacion(pieza).
```

Para establecer donde está cada objeto en el tiempo `T`, se usa el predicado `en`, de la siguiente manera.

```
en(pelota, oficina, 0). % la pelota está en la oficina en el tiempo 0
en(taza, cocina, 0).    % la taza está en la cocina en el tiempo 0
```

Para expresar que el agente tiene la pelota o la taza se utiliza el predicado `tiene`. De esta forma, cuando el agente tiene a la pelota en tiempo 4, esperamos que el modelo del programa contenga a `tiene(pelota,4)`.

Las acciones disponibles se describen a través del predicado `accion`, por ejemplo,

```
accion(mover_oficina_cocina). % describe la acción de moverse desde la oficina a la cocina
accion(mover_cocina_oficina). % describe la acción de moverse desde la cocina a la oficina
accion(recoger_pelota). % describe la acción de recoger la pelota
accion(dejar_pelota). % describe la acción de dejar la pelota
```

Suponiendo que se ocupa un predicado `exec`, tal que `exec(a,t)` representa que `a` se ejecuta en el tiempo `t`.

(a) **(2 puntos)** Escribe:

I. una regla que exprese que cuando el agente se mueve desde la oficina a la cocina en tiempo `T`, entonces en tiempo `T+1` el agentes se encuentra en la cocina.
Respuesta:

```
en(agente, cocina, T+1) :- exec(mover_oficina_cocina, T), en(agente, oficina, T).
```

II. una regla que exprese que cuando el agente se mueve desde la oficina a la cocina, la pelota sigue estando donde sea que estaba.
Respuesta:

```
en(pelota, X, T+1) :- exec(mover_oficina_cocina, T), en(pelota, X, T).
```

(b) **(2 puntos)** Escribe:

I. una regla que prohiba que el agente se pueda mover de la oficina a la cocina en tiempo `T` a menos que esté en la cocina en tiempo `T`.
Respuesta:

```
:- exec(mover_oficina_cocina, T), not en(agente, oficina, T).
```

II. una regla que prohibe al agente acarrear más de un objeto a la vez
Respuesta:

```
:- tiene(O1, T), tiene(O2,T), O1 != O2.
```

(c) **(2 puntos)** Escribe:

I. una regla que exprese que cuando el agente recoge la pelota, ahora el agente tiene la pelota.
Respuesta:

```
tiene(pelota, T+1) :- exec(recoger_pelota, T).
```

II. una regla que prohiba recoger la pelota a menos que el agente se encuentre en el lugar donde está la pelota.
Respuesta:

```
:- exec(recoger_pelota, T), en(agente, L, T), not en(pelota, L, T).
```

(d) **(2 puntos)** Escribe las reglas que permitan expresar el objetivo de que la pelota esté en la cocina en el tiempo 5.

Respuesta: Existen diversas formas de escribir esta respuesta. Acá 2 formas:

(A)
```
objetivo :- en(pelota, cocina, 5).
:- not objetivo.
```

(B)
```
:- not en(pelota, cocina, 5).
```

Pregunta 4 (6 puntos)

(a) **(3 puntos)** Abajo se muestran dos implementaciones del algoritmo de búsqueda genérica. Haz un argumento de uno en favor de otro cuando el algoritmo a implementar es BFS, utilizando como criterio el número de estados expandidos. Justifica a través de un ejemplo mostrando las diferencias en términos matemáticos, suponiendo un factor de ramificación b y que la solución se encuentra a profundidad p.

Input: Un problema de búsqueda $(\mathbf{S}, A, s_{init}, G)$
Output: Un nodo objetivo

1. *Open* es un contenedor vacío
2. *Closed* es un conjunto vacío
3. Inserta s_{init} a *Open*
4. $parent(s_{init}) = null$
5. **while** $Open \neq \emptyset$
6. $\quad u \leftarrow$ `Extraer`$(Open)$
7. $\quad$ Inserta u en *Closed*
8. $\quad$ **for each** $v \in Succ(u) \setminus (Open \cup Closed)$
9. $\quad\quad parent(v) = u$
10. $\quad\quad$ **if** $v \in G$ **return** v
11. $\quad\quad$ Inserta v a *Open*

Versión 1

Input: Un problema de búsqueda $(\mathbf{S}, A, s_{init}, G)$
Output: Un nodo objetivo

1. *Open* es un contenedor vacío
2. *Closed* es un conjunto vacío
3. Inserta s_{init} a *Open*
4. $parent(s_{init}) = null$
5. **while** $Open \neq \emptyset$
6. $\quad u \leftarrow$ `Extraer`$(Open)$
7. $\quad$ **if** $u \in G$ **return** u
8. $\quad$ Inserta u en *Closed*
9. $\quad$ **for each** $v \in Succ(u) \setminus (Open \cup Closed)$
10. $\quad\quad parent(v) = u$
11. $\quad\quad$ Inserta v a *Open*

Versión 2

Respuesta: El argumento es a favor de la Versión 1, que es la que vimos en clase. Cuando ese algoritmo se transforma en BFS (*Open* es una cola y por lo tanto se extrae el último que ha ingresado a la cola), Si la solución está a profundidad p el algoritmo descubre la solución desde su padre, es decir, cuando está expandiendo nodos a profundidad $p-1$. En ese momento, el número de estados expandidos es del orden de $b^p \approx 1 + b + b^2 + \ldots + b^{p-1}$. Sin embargo, la versión 2, para el mismo problema, retorna cuando ha llegado a expandir nodos de profundidad p. En este momento el algoritmo ha expandido b^{p+1} nodos, aproximadamente. El ahorro en expansiones de la versión 1 respecto de la 2 es aproximadamente $b^{p+1} - b^p = (b-1)b^p$.

(b) **(3 puntos)** Supón que ejecutas DFS sobre un árbol de profundidad 40, en donde cada nodo tiene exactamente 5 hijos. DFS es invocado desde la raíz del árbol y el problema de búsqueda **no tiene** solución.

I. ¿Cuántos elementos hay en Open, aproximadamente, cuando el algoritmo extrae el primer nodo que está a profundidad 10?
Respuesta: Si el factor de ramificación es b, DFS necesita almacenar del orden de $(b-1)p$ nodos en *Open* al llegar por primera vez a profundidad p. Es decir, aproximadamente 40 nodos. (Basta con una respuesta aproximada si el razonamiento es correcto)

II. ¿Cuántos elementos hay en Open, aproximadamente, cuando el algoritmo extrae el último nodo que está a profundidad 10?
Respuesta: Cuando se extrae el último elemento de *Open* a cierta profundidad ocurre que el algoritmo está revisando la última rama. Por tanto en *Open* solo existe 1 nodo.

III. ¿Cuántos nodos hay en Closed cuando la ejecución del algoritmo termina?
Respuesta: Cuando la ejecución termina, y dado que el problema no tiene solución, todos los nodos del árbol quedarán en closed; es decir $1 + 5 + 5^2 + \cdots + 5^{40} = 5^{41}/4$ (una respuesta aproximada con un razonamiento correcto será considerada también como correcta)