Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

PAUTA Interrogación 1

IIC2613 - Inteligencia Artificial

Segundo Semestre, 2019

Tiempo: 2 hrs.

1. *a*) Dé ejemplos de conocimiento de sentido común necesario para construir una máquina que pase el *Test de Turing*.

   R: Cualquier ejemplo tal como las sillas son para sentarse, las mesas para poner cosas.

   b) Escriba un programa clingo que represente el siguiente conocimiento. (a) *todo automovil es un vehículo* (b) *todo vehículo se puede desplazar entre dos ciudades* (c) *Santiago y Concepción son dos ciudades* (c) *R2R244 es un auto.*

   R:

   ```
   automovil(X) :- vehiculo(X).
   puedeDesplazar(V, X, Y) :- vehiculo(V), ciudad(X), ciudad(Y).
   ciudad(Santiago).
   ciudad(Concepcion).
   automovil(R2R244).
   ```

   c) Muestre cómo se instancian las reglas:

   ```
   1 {ejecuta(A,T) : accion(A), tiempo(T)} 1.
   1 {ejecuta(A,T) : accion(A)} 1 :- tiempo(T).
   ```

   (haga supuestos que considere necesario sobre cómo están definidos `accion` y `tiempo`)

   R: Como ejemplo veamos el siguiente programa

   ```
   accion(a1).
   accion(a2).
   tiempo(1..3).
   ```

   La primera regla va a instanciarse de modo que en cada modelo quede un par accion, tiempo. Es decir, en el ejemplo tendriamos 6 modelos, uno con cada par.

   En la segunda regla, se le esta pidiendo al programa que nos de todos los modelos en donde para cada tiempo, se ejecute solo una accion. Viendo el ejemplo, tendriamos modelos de la siguiente forma:

   ```
   ejecuta(a2,1) ejecuta(a1,2) ejecuta(a1,3)
   ejecuta(a2,1) ejecuta(a2,2) ejecuta(a1,3)
   ejecuta(a2,1) ejecuta(a1,2) ejecuta(a2,3)
   ejecuta(a2,1) ejecuta(a2,2) ejecuta(a2,3)
   ejecuta(a1,1) ejecuta(a1,2) ejecuta(a1,3)
   ejecuta(a1,1) ejecuta(a1,2) ejecuta(a2,3)
   ejecuta(a1,1) ejecuta(a2,2) ejecuta(a1,3)
   ejecuta(a1,1) ejecuta(a2,2) ejecuta(a2,3)
   ```

   d) Dé un programa con $5^n \cdot \binom{n}{4}$ modelos.

   R: Aqui sirve cualquier programa de 2 partes, donde la primera de 5^n modelos, y la segunda $\binom{n}{4}$, ya que al estar juntos, la cantidad de modelos se multiplica. Un ejemplo con $n = 5$ es:

```
q(1..5).
4 {p(X): q(X)} 4.
1 { r(I, 1); r(I, 2); r(I, 3); r(I, 4); r(I, 5) } 1 :- q(I).
```

e) Escriba un programa Π cuyo conjunto de términos instanciados sea $\{a, g(a), g(g(a)), \ldots\}$ tal que su versión instanciada (según la definición) sea *infinita*, y que posea un único modelo *finito*.

R: Un ejemplo es:

```
p(g(a)).
r(x) :- g(x).
```

Aquí se instancian infinitos terminos, pero su modelo es solo p(g(a)).

No sirve poner:

```
a.
g(X) :- X.
```

Ya que la X no puede ir como predicado en un programa

f) Calcule cuántos modelos tiene el siguiente programa.

```
p(X) :- r(X), not q(X).
q(X) :- r(X), not p(X).
r(1..3).
```

R: El programa tiene $2^3 = 8$ modelos.

2. El problema de planificar los movimientos de un robots al interior de un edificio generalmente se reduce al de encontrar un camino en una grilla. A su vez, el problema de encontrar un camino en una grilla se puede representar como un problema de búsqueda.

   La Figura 1 muestra un estado inicial, en donde el robot está en la posición $(1, 1)$. El robot debe llegar a $(3, 2)$. Las acciones posibles son *arriba*, *abajo*, *izquierda*, *derecha*. El robot puede ejecutar cualquier acción que lo lleve a una celda adyacente sin atravesar un muro. Los muros representados por la líneas gruesas entre celdas.

   a) **(0,5 puntos)** Diga cómo representaría un estado del espacio de búsqueda de este problema (por ejemplo, en Python).

   R: Un estado puede ser representado mediante las coordenadas de la posición en la que se encuentra. En python, podría representarse con una tupla (x, y).

   b) **(2 puntos)** Diga (o dibuje) qué estados quedan en Closed y en Open justo antes de que retorne una ejecución DFS. Suponga que DFS agrega estados a la pila (*stack*) Open en el siguiente orden: primero el que resulta de ejecutar *arriba*, segundo el que resulta de ejecutar *derecha*, tercero el que resulta de ejecutar *abajo*, cuarto el que resulta de ejecutar *izquierda*. (Por favor, vuelva a leer la oración anterior y asegúrese que la entiende antes de continuar.)

   R: Un nodo entra a OPEN al momento de ser generado. Un nodo entra a CLOSE al momento de ser expandido (generar sucesores). En esta ocasión, se asume que se revisa si el nodo es GOAL al momento de ser generado

   Open : {(4,3), (3,4), (2,4), (1,2)} **Closed :** {(1,1), (2,1), (2,2), (2,3), (1,3), (1,4), (3,3)}

   c) **(2 puntos)** Diga (o dibuje) qué estados quedan en Closed y en Open justo antes de que retorne una ejecución de BFS. Suponga que BFS agrega estados a la cola Open en el siguiente orden: primero el que resulta de ejecutar *arriba*, segundo el que resulta de ejecutar *derecha*, tercero el que resulta de ejecutar *abajo*, cuarto el que resulta de ejecutar *izquierda*.

   R: Open : {(4,3), (3,4), (1,3) } **Closed :** {(1,1), (1,2), (2,1), (2,2), (2,3), (2,4), (3,3)}

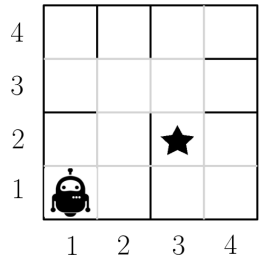

Figura 1: Situación inicial del mundo de navegación usada en preguntas a), b), y c).

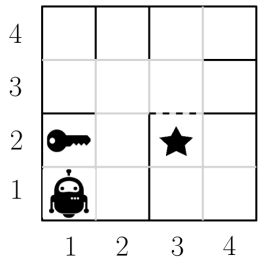

Figura 2: Situación inicial del mundo de navegación con llave. Usada solo en pregunta d).

d) **(1,5 puntos)** Suponga ahora que el problema es extendido con una llave, y que se agrega una acción adicional, *recoger*, que hace que el robot recoja la llave. Un posible problema se ilustra en la Figura 2. El robot solo puede atravesar de $(3,3)$ a $(3,2)$ si es que ha recogido la llave. Diga ahora cómo se define un estado y calcule cuántos estados hay en este nuevo espacio de búsqueda.

R: Un estado se representa por la posición, pero en este caso también es importante si es que tiene la llave. Ante esto, el estado sería representado con tres parámetros: $(x, y, tiene\_llave)$, siendo $tiene\_llave$ un booleano. Por lo tanto, el número de estados aumenta al doble: 32 estados posibles.

3. Suponga que para representar n grafos no dirigidos en un mismo programa clingo se utilizan los siguientes predicados:

   - `grafo(N)`: expresa que `N` es un grafo del programa.
   - `nodo(U,N)`: que expresa que `U` es un nodo del grafo número N.
   - `arco(U,V)`: que expresa que `(U,V)` es un arco. Suponemos que en el programa existe la regla:

     ```
     arco(X,Y) :- arco(Y,X).
     ```

     y, además, que nunca un usuario declarará un nodo como parte de dos grafos distintos, ni tampoco declarará arcos entre nodos de grafos distintos.

   De esta manera, para definir dos grafos, un usuario podría escribir el siguiente código.

```
% definición del primer grafo
grafo(1).
nodo(a,1).
nodo(b,1).
nodo(c,1).
arco(a,b).
```

```
arco(b,c).
arco(a,c).

% definición del segundo grafo
grafo(2).
nodo(u,2).
nodo(v,2).
nodo(w,2).
arco(u,v).
arco(v,w).

arco(X,Y) :- arco(Y,X). % esta regla está incluida siempre por defecto
```

a) **(2 puntos)** Escriba el predicado `conexo`, tal que `conexo(N)` es está en un modelo del programa si y solo si el grafo `N` es conexo. (Recuerde que un grafo es conexo cuando existe un camino entre cada par de nodos del grafo.)

b) **(4 puntos)** Intuitivamente, dos grafos G y H son *isomorfos* cuando son estructuralmente iguales; es decir, es posible dibujar ambos grafos de tal manera que se vean idénticos.

Matemáticamente, dos grafos $G = (V_G, E_G)$ y $H = (V_H, E_H)$ son isomorfos si existe una relación R uno-a-uno (o biyectiva) entre nodos de G y H tal que si $(a, u) \in R$ y $(b, v) \in R$ entonces $\{a, b\} \in E_G$ ssi $\{u, v\} \in E_H$. Finalmente, una relación biyectiva R entre dos conjuntos A y B es una que: (1) para todo $a \in A$, existe un único $b \in B$ tal que $(a, b) \in R$ y (2) para todo $b \in B$, existe un único $a \in A$ tal que $(a, b) \in R$.

Escriba reglas que permitan obtener uno o más modelos que describan todos los isomorfismos que existen entre los grafos 1 y 2. Si los grafos 1 y 2 no son isomorfos, el programa resultante no debe tener modelos.

i. Dos restricciones cardinalidad. La primera expresando que para cada nodo `A` del grafo 1, existe exactamente un nodo `B` del grafo 2 tal que se cumple que `r(A,B)`. La otra restricción de cardinalidad expresa lo inverso: para cada nodo `B` del grafo 2, hay exactamente un nodo `A` del grafo 1 tal que se cumple que `r(A,B)`.

ii. Una o más restricciones, que usen los predicados `nodo`, `arco`, y `r`, que finalmente definan la condición de isomorfismo que debe cumplir `r`.