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

Repaso Control Largo 1

1. ¿Cuántos nodos, aproximadamente, se almacenan en la lista Open del algoritmo DFS en el momento que se expande un nodo a profundidad p? Haga todos los supuestos que permitan simplificar (sin trivializar) su respuesta.

   Respuesta: Suponiendo que el factor de ramificación del árbol de DFS es b, entonces, por cada nivel del árbol hay (b - 1) o menos nodos. Entonces podemos aproximar el número por p(b - 1) (o bp).

2. Describa un programa que tenga $\binom{n}{5}$ modelos.

   Respuesta: El programa contiene las reglas $p(a_1), p(a_2), ..., p(a_n)$. Más la regla $5q(X) : p(X)5$.

3. Describa un programa que tenga 5^n modelos.

   Respuesta: Acá usamos n predicados: $p_1, ..., p_n$. El programa contiene, para cada $p_i(i \in 1, ..., n)$, la restricción de cardinalidad.

4. Suponga que usa el predicado *nodo* y *arco* para definir los nodos y arcos de un grafo dirigido. Escriba reglas que permitan deducir cuáles son los nodos *superalcanzables*, que son aquellos que son alcanzables desde todo nodo del grafo.

   Respuesta:

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

   % un nodo no es superalcanzable si desde alg´un nodo Y no es posible alcanzar a X
   nosuperalcanzable(X) :- nodo(Y),not alcanzable(Y,X).

   % Un nodo es super alcanzable cuando no es posible demostrar que no es nosuperalcanzable
   superalcanzable(X) :- not nosuperalcanzable(X).
   ```

5. Los circuitos digitales tienen muchísimas aplicaciones prácticas. En esta pregunta exploraremos cómo modelar un circuito digital compuesto por componentes lógicas Booleanas. En nuestra modelación supondremos que cada cable puede transmitir una señal binaria (0 o 1), y utilizaremos solamente los siguientes predicados:

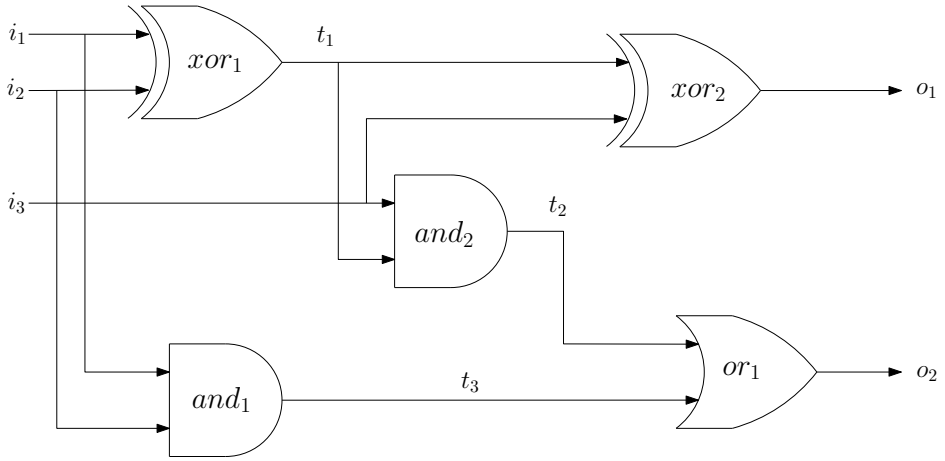
   - $salida(t, x, y, z)$: expresa que la componente de tipo t, cuando sus entradas son x e y, tiene como salida z.
   - $cable(c)$: expresa el hecho que c es un cable.
   - $componente(m)$: expresa el hecho que m es una componente.
   - $val(c, v)$: expresa el hecho la señal que transmite el cable c es v.
   - $defectuosa(m)$: expresa que la componente m está defectuosa.

   Parte de las reglas del programa para modelar el circuito se encuentran ya escritas en el reverso de esta hoja.

a) Escriba 5 reglas adicionales que permitan deducir qué valor está asociado a los cables $t1$, $t2$, $t3$, $o1$, y $o2$, en el caso que las componentes **no sean defectuosas**. Cada regla debe mencionar (correctamente) al predicado *defectuosa* en el lado derecho. En particular, si a su programa se agregan las tres sentencias $val(i1, 1)$, $val(i2, 0)$, $val(i3, 1)$, entonces se debe deducir que $val(o1, 0)$ y $val(o2, 1)$.

b) Ahora supondremos que cuando una componente m falla, entonces m podría dar cualquier salida, tanto 0 como 1, independiente de cuál es la entrada. Escriba 5 reglas adicionales—una por componente—que modelen el comportamiento del circuito cuando las componentes puedan fallar.

c) ¿Cómo es posible usar el sistema de programación en lógica para descubrir cuál componente está fallando si ante la entrada descrita por $val(i1, 1)$, $val(i2, 0)$, $val(i3, 1)$, se obtiene un 1 en $o1$ y un 0 en $o2$? Diga qué agregar al programa y luego cómo obtener la respuesta.




```
salida(xor,0,0,0).
salida(xor,0,1,1).
salida(xor,1,0,1).
salida(xor,1,1,0).

salida(or,0,0,0).
salida(or,0,1,1).
salida(or,1,0,1).
salida(or,1,1,1).

salida(and,0,0,0).
salida(and,0,1,0).
salida(and,1,0,0).
salida(and,1,1,1).

cable(i1).
cable(i2).
cable(i3).
cable(t1).
cable(t2).
cable(t3).

componente(xor1).
```

```
componente(xor2).
componente(and1).
componente(and2).
componente(or1).
```

Respuetas:

```
%% PARTE a)
val(t1,V) :- salida(xor,V1,V2,V),val(i1,V1),val(i2,V2),not defectuosa(xor1).
val(o1,V) :- salida(xor,V1,V2,V),val(t1,V1),val(i3,V2),not defectuosa(xor2).
val(t3,V) :- salida(and,V1,V2,V),val(i1,V1),val(i2,V2),not defectuosa(and1).
val(t2,V) :- salida(and,V1,V2,V),val(i3,V1),val(t1,V2),not defectuosa(and2).
val(o2,V) :- salida(or,V1,V2,V),val(t2,V1),val(t3,V2),not defectuosa(or1).

%% PARTE b)
val(t1,0); val(t1,1) :- val(i1,V1),val(i2,V2),defectuosa(xor1).
val(o1,0); val(o1,1) :- val(t1,V1),val(i3,V2),defectuosa(xor2).
val(t3,0); val(t3,1) :- val(i1,V1),val(i2,V2),defectuosa(and1).
val(t2,0); val(t2,1) :- val(i3,V1),val(t1,V2),defectuosa(and2).
val(o2,0); val(o2,1) :- val(t2,V1),val(t3,V2),defectuosa(or1).

%% PARTE c)
1 {defectuosa(C) : componente(C)} 1.
observacion :- val(o1,1),val(o2,0).
:- not observacion.
```

6. 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.*

7. 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`)

8. 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).

   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.)

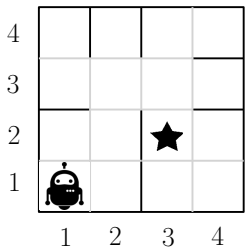


Figura 1: Situación inicial del mundo de navegación usada en preguntas a), b), y c).

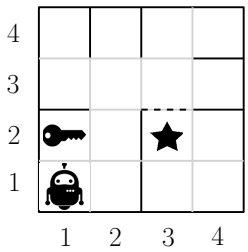


Figura 2: Situación inicial del mundo de navegación con llave. Usada solo en pregunta d).

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*.

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.

9. 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`.

10. (2 puntos) Al "relajar" un problema de búsqueda P, se obtiene un problema de búsqueda P' cuyo espacio de estados y grafo de búsqueda están íntimamente relacionados al de P. Explique en detalle cuál es esta relación, ilustre con un ejemplo de un problema de búsqueda, y demuestre que si c es la solución óptima a P y c' es la solución óptima para P', entonces $c' \leq c$.

11. El algoritmo A*, retorna un nodo objetivo inmediatamente después de extraerlo desde la OPEN. Considere una versión de A* que, en vez de hacer aquello, retorna un nodo objetivo inmediatamente después de que este es generado (y antes de ser agregado a la lista OPEN).

a) (2 puntos) Muestre que esta variante de A* es subóptima. Use como contraejemplo un espacio de búsqueda con tres estados.

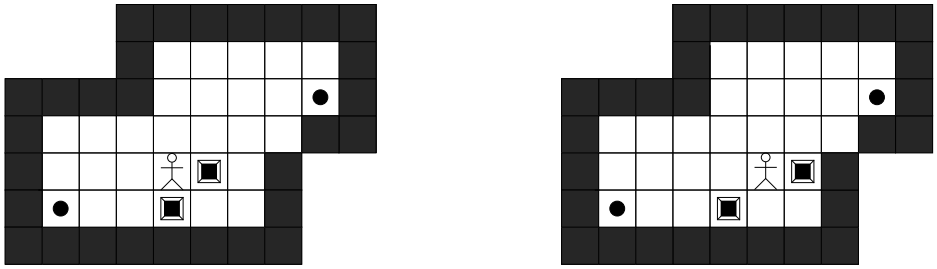
b) (2 puntos) Sea c_{sol} el costo de la solución encontrada por esta variante de A* y suponga se utiliza una heurística admisible. Demuestre, que $c^* \leq c_{\text{sol}} \leq c^* + c_{\text{ult}}$, donde c^* es el costo de la solución óptima al problema de búsqueda en cuestión y c_{ult} es el costo del último arco del camino que conecta el estado inicial con el estado retornado.

12. El juego de *Sokoban* es un puzle cuyo objetivo es empujar cajas hasta posiciones de almacenamiento. En este problema, un agente puede moverse sobre una grilla hacia arriba, abajo, a la izquierda o

a la derecha. Una posición de la grilla puede estar bloqueada (*obstáculo*), puede contener una caja, o puede contener un punto de destino. En una misma celda, no pueden haber dos cajas simultáneamente, ni tampoco estar el agente y una caja.

El agente puede navegar sobre cualquier posición de la grilla sin obstáculos. Al avanzar el agente sobre una posición que contiene una caja, ésta se desplaza una celda en la dirección del movimiento del agente, a menos que haya un obstáculo en tal celda, en cuyo caso el movimiento del agente no es posible.

La figura muestra un estado particular y el resultado de ejecutar un movimiento a la *derecha*.



a) Sea $d_m(c_1, c_2)$ la distancia Manhattan entre las celdas c_1 y c_2. Defina una heurística admisible en base a sumas de distancias manhattan que retorne valor no nulo para todo estado que no es un objetivo. Demuestre que la heurística es admisible.

b) Sea h_m la heurística anterior. ¿Es $2 \cdot h_m$ admisible? Demuestre su respuesta.

c) Dé una heurística admisible mejor que la definida en *b*), que sea computable en tiempo polinomial en el tamaño de la grilla. Justifique que es admisible.

13. (*Answer Set Programming*) En esta pregunta buscamos comprender cómo desarrollar un agente inteligente capaz de jugar al juego *Buscaminas* (*Minesweeper*, en inglés), usando la herramienta de programación en lógica. El Buscaminas se juega en un tablero rectangular, en donde hay una cantidad desconocida de celdas que contienen una *mina*. Cada celda que contiene una mina se encuentra además *oculta*. Existe un número de celdas que están ocultas pero que no contienen una mina. Cada celda que no está oculta puede desplegar un número, que indica cuántas de sus celdas *vecinas* contienen minas. Si una celda no oculta no despliega un número, significa que ninguna de sus celdas vecinas contiene una mina. Cada celda tiene como vecinas a las ocho celdas adyacentes. La siguiente figura muestra un posible tablero.

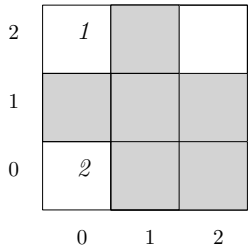


Figura 3: Un tablero de buscaminas de 3×3. Las celdas no ocultas son $(0, 0)$, $(0, 2)$ y $(2, 2)$.

Para representar el problema usando programación en lógica, se utilizan los siguientes predicados.

- $celda(x, y)$ expresa que (x, y) es una celda del tablero.
- $vecina(x, y, x', y')$ expresa que (x, y) es vecina de (x', y').
- $oculta(x, y)$ expresa que la celda (x, y) está oculta.

- $libre(x, y)$ expresa que la celda (x, y) está libre, es decir, no contiene una mina.
- $mina(x, y)$ expresa que la celda (x, y) contiene una mina.

Conteste las siguientes preguntas (1 punto por pregunta):

a) Escriba reglas que definan el predicado *vecina*.

Respuesta:
```
nulas(0,0).
inc(-1,0,1).
vecina(X,Y,X+Dx,Y+Dy) :- celda(X+Dx,Y+Dy), inc(Dx), inc(Dy), not nulas(Dx,Dy).
```

b) Escriba una regla que exprese el hecho que cada celda del tablero está libre o contiene una mina.

Respuesta:
```
1 {libre(X,Y);mina(X,Y)} 1 :- celda(X,Y).
```

c) Escriba reglas que expresen que:

1) Hay exactamente dos minas en las celdas vecinas a $(0, 0)$.
   Respuesta:
   ```
   2 {mina(X,Y) : vecina(0,0,X,Y)} 2
   ```
2) Hay exactamente una mina en las celdas vecinas a $(0, 2)$.
   Respuesta:
   ```
   1 {mina(X,Y) : vecina(0,2,X,Y)} 1
   ```

d) Explique detalladamente cómo haría usted para, usando un solver como clingo, obtener dónde jugar sin riesgo de que la celda explote (por la presencia de una mina).

Respuesta: Computo todos los modelos usando el solver. Si todos los modelos coinciden en que $libre(x, y)$ para cierta celda (x, y) que está oculta, entonces podemos jugar ahí.

e) Después de hacer una jugada, se revelan nuevos números en el tablero. Explique qué debe *agregar* y *eliminar* al programa para poder tomar la siguiente decisión.

Respuesta: Al jugar el tablero despliega más números. Basta con agregar reglas como las de la pregunta c) para representar la información de esas reglas. Es necesario eliminar la regla $oculta(x, y)$ donde (x, y) es la celda donde recién se jugó.

14. Una heurística si dice *consistente* si y solo si:

    i. $h(s) \leq c(s,t) + h(t)$ para todo estado s y todo estado t que es sucesor de s, donde $c(s,t)$ representa el costo de la acción que lleva desde s a t, y

    ii. $h(s_g) = 0$ para todo estado objetivo s_g

    Conteste las siguientes preguntas:

    a) (2/3 del puntaje) Demuestre que si h es consistente, cada vez que A* expade un nodo s, todo t que es sucesor de s y que es agregado a Open en esa misma iteración es tal que $f(t) \geq f(s)$.
    Respuesta: Sumando $g(s)$ a ambos lados obtenemos:

    $$g(s) + h(s) \leq g(s) + c(s,t) + h(t)$$

    Si t es agregado a Open entonces $g(t) = g(s) + c(s,t)$ por lo que podemos reemplazar arriba y obtener $g(s) + g(s) \leq g(t) + h(t)$ que es equivalente a lo que se quiere mostrar.

    b) (1/3 del puntaje) Diga por qué esta relación podría no cumplirse con algunos sucesores de s que ya estaban en Open.
    Respuesta: Cuando un sucesor de s, digamos t, no se agrega a Open, entonces se da que $g(s) + c(s,t) \geq g(t)$, por esta razón la desigualdad de arriba podría perfectamente invalidarse de hacer la sustitución que hicimos arriba.

15. Los problemas de búsqueda determinísticos pueden ser descritos por un estado inicial, un conjunto de acciones con precondiciones y efectos, y una condición objetivo. Una forma de lograr esto es definir los estados como un conjunto de átomos. Por ejemplo un estado inicial podría ser $S_{init} = \{en(sala), soleado(hoy)\}$. Las acciones, a su vez, están definidas por una precondición y un efecto. Formalmente cada acción a tiene una precondición $pre(a)$, un efecto positivo $add(a)$ y un efecto negativo $del(a)$. Esto permite definir el estado sucesor usando simple aritmética de conjuntos. Si s es un estado, a es ejecutable en s si y solo si $pre(a) \subseteq s$. Si a es ejecutable en s, el estado resultante queda dado por $(s \cup add(a)) \setminus del(a)$.

    Por ejemplo, la acción $caminar(sala, metro)$ podría tener la precondición $\{en(sala)\}$, el efecto positivo $\{en(metro)\}$ y el efecto negativo $\{en(sala)\}$. Así, $caminar(sala, metro)$ es aplicable en S_{init} (porque $\{en(sala)\} \subseteq S_{init}$) al aplicar esta acción sobre S_{init}, obtendríamos el estado $\{en(metro), soleado(hoy)\}$.

    Finalmente, la condición objetivo G se define como un conjunto de átomos que queremos que se cumplan. De esta forma, si llegamos a un estado que sea superconjunto de G, habremos logrado nuestro objetivo. Por ejemplo para decir que nuestro objetivo es estar en la piscina, podríamos $G = \{en(piscina)\}$. En otras palabras, cualquier estado que contenta el átomo $en(piscina)$ es un estado objetivo.

    Sea P un problema de búsqueda definido de la manera descrita arriba.

    a) Sea P^{pre} el problema que resulta de cambiar las precondiciones de toda acción de P por el conjunto vacío y dejar los otros elementos intactos. Justifique que P^{pre} es una relajación del problema P.
    Respuesta: Al elimnar las precondiciones toda acción es aplicable en todo estado. El grafo del espacio de estados queda con más estados y más arcos. Esto garantiza que todo camino que existía en el grafo original ya también existe en el nuevo grafo. Por lo tanto, esta es una relajación al problema.

    b) Sea P^{-} el problema que resulta cambiar los efectos negativos de todas las acciones de P por el conjunto vacío y dejar los otros elementos intactos. Justifique que P^{-} es una relajación del problema P.

Respuesta: Al eliminar los efectos negativos cualquier solución que antes resolvía el problema también lo resuelve ahora. En ese sentido, funciona como una relajación.

c) Suponga que para obtener una heurística para un estado s resolvemos el problema relajado desde s y luego retornamos el costo de la solución obtenida. Diga por qué usar P^-, en general, permite obtener mucha más información que usar P^{pre}.

Respuesta: Con P^{pre} se obtiene muy poca información, porque para todo problema hay acciones cuyos efectos resuelven el problema de inmediato. La heurística siempre sería igual al número de acciones necesario para cumplir cada objetivo en G. En el ejemplo $lanzarse(piscina)$ tendría como efecto $en(piscina)$. Al elimnar precondiciones esta acción es siempre posible en todo estado, por lo que con la relajación siempre se encuentra que $h = 1$. El efecto que esto tiene es que la heurística discrimina muy poco entre pares de estados.