Solución - Evaluación (1ª instancia)

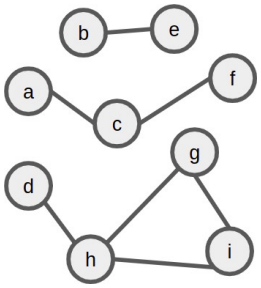
Duración: 3 horas

Ejercicio 0: Pregunta individual sobre el laboratorio

Describa la heurística utilizada para evaluar el tablero de juego en su estrategia minimax.

Ejercicio 1 [30 puntos]

Considere el problema de encontrar las componentes conexas de un grafo. Los grafos con los que trataremos son no dirigidos y pueden contener ciclos. Por ejemplo, el siguiente grafo tendría tres componentes conexas: [a,c,f], [b,e], y [d,g,h,i].



El grafo está implementado en la base de cláusulas de prolog mediante los predicados vertice/1 y arista/2. Por ejemplo:

vertice(a).	arista(a, c).	Observación: el grafo es no dirigido, pero se definirá
vertice(b).	arista(b, e).	una arista como par de vértices una sola vez, o sea
vertice(c).	arista(c, f).	que si aparece <g,h> no aparecerá <h,g>
vertice(d).	arista(d, h).	
...	...	

Una solución al problema es una lista de componentes, donde cada componente es una lista de vértices no repetidos. En este caso una solución sería:

[[a,c,f], [b,e], [d,g,h,i]]

Implemente en Prolog los siguientes predicados:

a) camino(+X,?Y) ← Existe un camino entre X e Y. Notar que se deben cumplir las condiciones indicadas anteriormente: las aristas se pueden recorrer en ambos sentidos y el grafo puede contener ciclos, por lo que hay que tener cuidado con esto en el programa.

```
camino(a,f).

camino(g,d).

camino(b,e).

camino(e,b).
```

b) `elegir_no_visitado(+Componentes,?V)` ← V es un vértice del grafo que no pertenece a ninguna de las componentes que están en la lista Componentes. Por ejemplo:

```
elegir_no_visitado([],c).

elegir_no_visitado([[a,c,f]],b).

elegir_no_visitado([[a,c,f],[b,e]],h).
```

c) `componente_conexa(+V,?C)` ← C es la componente conexa a la que pertenece el vértice V. Recordar que la componente conexa debe ser una lista sin elementos repetidos. Por ejemplo:

```
componente_conexa(g,[g,d,h,i]).
```

d) `componentes_conexas(?C)` ← C es la lista de todas las componentes conexas del grafo.

Solución

a)

```
arista_direccion(X,Y):-arista(X,Y).
arista_direccion(X,Y):-arista(Y,X).


camino(X,Y):-camino(X,Y,[]).
camino(X,Y,Visitados):-arista_direccion(X,Y),\+member(Y,Visitados).
camino(X,Y,Visitados):-arista_direccion(X,Z),\+member(Z,Visitados),camino(Z,Y,[Z|
Visitados]).
```

b)

```
member_subset([S|_],X):-
    member(X,S).
member_subset([_|Rest],X):-
    member_subset(Rest,X).


elegir_no_visitado(Componentes,V):-
    vertice(V),
    \+member_subset(Componentes,V).
```

c)

```
componente_conexa(V,C):-
    setof(W,camino(V,W),C).
```

d)

```
componentes_conexas(C):-
    componentes_conexas([],C).


componentes_conexas(Componentes,Componentes2):-
    elegir_no_visitado(Componentes,V),!,
    componente_conexa(V,C),
    componentes_conexas([C|Componentes],Componentes2).
componentes_conexas(Componentes,Componentes).
```

Ejercicio 2 [15 puntos]

a) Indique si son verdaderas o falsas las siguientes afirmaciones. Fundamente.

i) Un objetivo definido contiene uno y solo un literal positivo.
ii) Pueden existir muchos unificadores diferentes para un par de expresiones.
iii) Dado un programa lógico y un objetivo, el árbol SLD resultante siempre es finito.
iv) Un árbol SLD contiene todas las respuestas correctas para una consulta.

b) Aplique el algoritmo para hallar el m.g.u. de las siguientes dos expresiones, mostrando su aplicación paso a paso:

```
pred(p(Y,X),h(Y),m(W,X))      pred(Z,T,m(Z,T))
```

Solución

a)
i) Falso: Por definición, un objetivo definido no tienen ningún literal positivo.
ii) Verdadero: Por ejemplo p(X) y p(Y) tienen como unificadores {X/a,Y/a} o {X/b, Y/b}.
iii) Falso: Pueden existir ramas infinitas en un árbol SLD.
iv) Falso: Pueden el árbol contiene las respuestas computadas, pero puede haber respuestas correctas que no son computadas.

b)

```
stack={ pred(p(Y,X),h(Y),m(W,X)) = pred(Z,T,m(Z,T)) }      mu={}
caso 4
stack={ p(Y,X)=Z, h(Y)=T, m(W,X)=m(Z,T)) }                 mu={}
caso 2
stack={ h(Y)=T, m(W,X)=m(p(Y,X),T)) }                      mu={ Z/p(Y,X) }
caso 2
stack={ m(W,X)=m(p(Y,X),h(Y))) }                           mu={ Z/p(Y,X), T/h(Y) }
caso 4
stack={ W=p(Y,X), X=h(Y) }                                 mu={ Z/p(Y,X), T/h(Y) }
caso 1
stack={ X=h(Y) }                                  mu={ Z/p(Y,X), T/h(Y), W/p(Y,X) }
caso 1
stack={  }                          mu={ Z/p(Y,h(Y)), T/h(Y), W/p(Y,h(Y)), X/h(Y) }
fin, resultado: mu={ Z/p(Y,h(Y)), T/h(Y), W/p(Y,h(Y)), X/h(Y) }
```

Ejercicio 3 [25 puntos]

Considere el siguiente programa Prolog:

```
a(1).
a(2).
a(X) :- p(X,1).

b(3).
b(4).
c(3).
d(4).
e(4).

p(X,Y) :- b(X), c(X), d(Y), e(Y).
p(X,Y) :- a(X),  b(Y).
p(7,8).
```

a) Dibuje el árbol SLD correspondiente a la consulta **`?- p(X,Y)`** suponiendo que la regla de computación toma el átomo de más a la izquierda para la resolución.
b) ¿Qué respuestas dará el intérprete Prolog para el objetivo anterior?
c) ¿Cuáles son las respuestas si sustituimos la segunda cláusula del predicado `p/2` por la siguiente? Justifique.

```
p(X,Y) :- a(X), !,  b(Y).
```

d) ¿Cuáles son las respuestas si eliminamos del programa original la cláusula `a(X) :- p(X,1).`? Justifique.

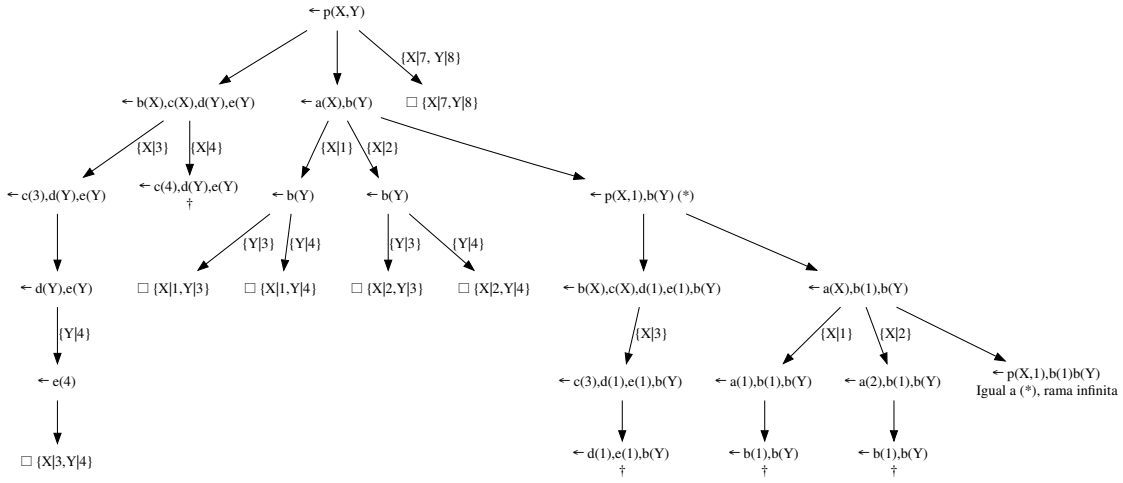
e) Construya una consulta Prolog que devuelva en una lista todas las respuestas a la consulta `p(X,Y)`, con el programa del punto d).

f) Escriba el siguiente predicado en Prolog:

`exOr(+Ob1, +Ob2)` ← El predicado es verdadero si solamente uno de los dos objetivos, `Ob1` u `Ob2`, es verdadero.

Solución

a)



b)

```
?- p(X,Y).
X = 3,
Y = 4 ;
X = 1,
Y = 3 ;
X = 1,
Y = 4 ;
X = 2,
Y = 3 ;
X = 2,
Y = 4 ;
(falla por límite de stack)
```

c)

```
?- p(X,Y).
X = 3,
Y = 4 ;
X = 1,
Y = 3 ;
X = 1,
Y = 4.
```

d)

```
?- p(X,Y).
X = 3,
Y = 4 ;
X = 1,
Y = 3 ;
X = 1,
Y = 4 ;
X = 2,
Y = 3 ;
X = 2,
Y = 4 ;
X = 7,
Y = 8.
```

(Se elimina la rama infinita, y por lo tanto se encuentra la última solución)

e)

```
?- findall((X,Y),p(X,Y),L).
L = [(3, 4),  (1, 3),  (1, 4),  (2, 3),  (2, 4),  (7, 8)].
```

f)

```
exOr(Obj1,Obj2) :- Obj1, Obj2,!,fail.
exOr(Obj1,Obj2) :- Obj1,!.
exOr(Obj1,Obj2) :- Obj2.
```

Ejercicio 4 [15 puntos]

a) Considere un escenario donde existe una moneda desbalanceada (la probabilidad de que salga Cara es de un 40%), y dos urnas con bolas: en la primera, hay 70 bolas rojas y 30 azules, mientras que en la segunda hay 50 bolas azules, 30 rojas y 20 negras. Un jugador lanza la moneda y saca una bola al azar de cada una de las urnas, y gana si la moneda sale cara y al menos una bola es azul, o si las dos bolas son del mismo color.

Construya un programa en Problog que permita saber la probabilidad de ganar el juego.

b) Utilizando DCG, define un programa Prolog para reconocer el siguiente lenguaje:

$L = \{a^n\ b^m\ c^{m+n}\ d^{m*n} / n,m \geq 0\}$

Solución

a)

```
% Moneda
0.4 :: cara.

% Urnas
0.7 :: urna(1,roja) ; 0.3 :: urna(1,azul).
0.5 :: urna(2,azul) ; 0.3 :: urna(2,roja) ; 0.2 :: urna(3,negra).

% Predicado para la victoria
win :- cara, urna(_,azul).
win :- urna(1,Color), urna(2,Color).

query(win).
```

b)

```
s --> a(N),b(M),{NmasM is N + M, NporM is N * M},c(NmasM),d(NporM).
a(0) --> [].
a(N1) --> [a],a(N),{N1 is N + 1}.
b(0) --> [].
b(N1) --> [b],b(N),{N1 is N + 1}.
c(0) --> [].
c(N1) --> [c],c(N),{N1 is N + 1}.
d(0) --> [].
d(N1) --> [d],d(N),{N1 is N + 1}.
```