Evaluación (junio 2016)

Duración: 3 horas

Ejercicio 1 [evaluación individual del laboratorio]

(Individual)

Ejercicio 2 [15 puntos]

a) Dé una interpretación que sea modelo y una que no lo sea para los siguientes programas lógicos:

```
 i.    p(f(X),Y) <- p(X,Y).
       p(X,X).
       q <- s.

ii.    p(X,s(X,Xs)).
       p(X,s(Y,Ys)) <- p(X,Y).
```

b) Para el programa `i`, considere el objetivo `<-p(X,Y)` y la sustitución `{X|f(4),Y|4}`. ¿Es una respuesta correcta? ¿Es una respuesta computada?

a)

i.
Sean las interpretaciones I1 e I2, ambas con dominio D = Nat, Mf= suc(x), q y s verdaderas. En I1, además, p se interpreta por la relación ≥, y I1 es entonces un modelo de ii. Para I2, basta con interpretar p con la relación < para que la interpretación resultante no sea modelo.

ii.
Sean las interpretaciones I1 e I2, ambas con dominio D = {A,B}, y mapeo de s según la siguiente tabla :

X Y	s (X,Y)
A A	A
A B	A
B A	B
B B	B

Basta con interpretar p con la relación '=' en D (I1) para obtener un modelo y p con la relación '≠' para obtener una interpretación que no es modelo (I2).

b)
Es una respuesta correcta, ya que el programa implica lógicamente la clausura universal del objetivo con la sustitución aplicada. No es una respuesta computada, ya que es demasiado específica y la resolución computa siempre el mgu.

Ejercicio 3 [23 puntos]

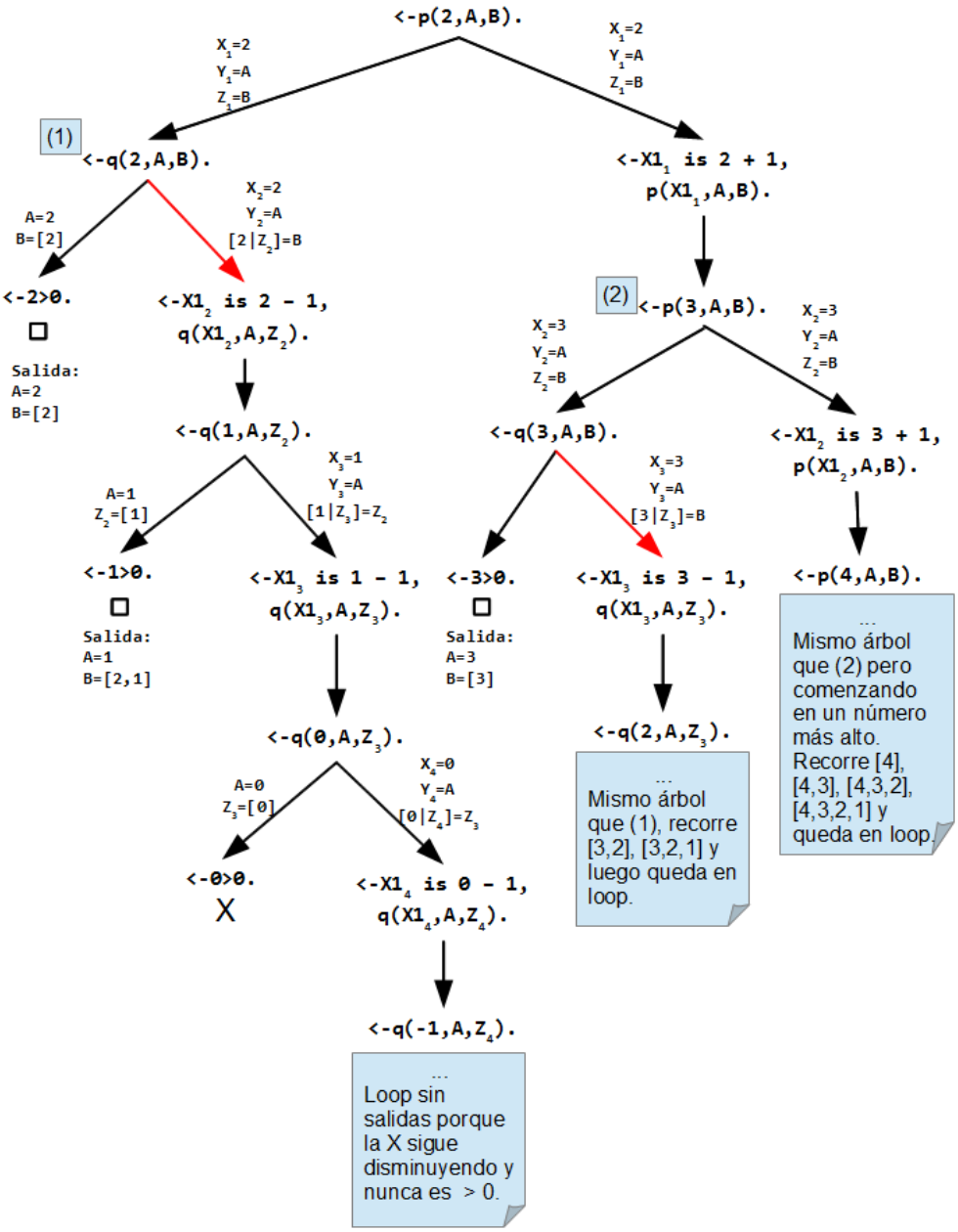
Sea el siguiente programa lógico P:

```
p(X,Y,Z):-
    q(X,Y,Z).
p(X,Y,Z):-
    X1 is X + 1,
    p(X1,Y,Z).

q(X,X,[X]):-
    X > 0.
q(X,Y,[X|Z]):-
    X1 is X - 1,
    q(X1,Y,Z).
```

a) Construya el árbol SLD correspondiente al objetivo `<-p(2,A,B)` asumiendo que el intérprete selecciona el átomo de más a la izquierda como regla de computación.

Solución:



b) Indique qué respuestas dará el intérprete de Prolog para el objetivo anterior.
Solución:
A=2, B=[2]
A=1, B=[2,1]
Luego entrará en loop sin devolver ninguna respuesta.

c) Indique qué respuestas dará el interprete de Prolog si se agrega un cut en la primer regla del predicado q/3 de la siguiente manera:

```
q(X,X,[X]):-
    X > 0, !.
```

Solución:
Se podan las ramas marcadas en rojo en el árbol (y no recorre la rama infinita que produce el loop). Por lo tanto las salidas que se devuelven son:
A=2, B=[2]
A=3, B=[3]

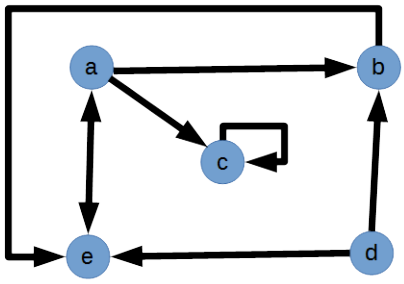
A=4, B=[4]
...

Ejercicio 4 [35 puntos]

Considere un grafo dirigido representado mediante el predicado arista. Por ejemplo, si se tiene el siguiente programa:

```
arista(a, b).
arista(a, c).
arista(a, e).
arista(b, e).
arista(c, c).
arista(d, b).
arista(d, e).
arista(e, a).
```

se estaría representando el grafo de la figura de la derecha.



Implementar los siguientes predicados, implementando los predicados auxiliares que se usen (excepto los de segundo orden):

a) `matriz_adyacencia(+LV, ?M) <-` M es la matriz de adyacencia del grafo representado en el programa mediante el predicado arista, para la lista de vértices LV. Cada celda (I, J) tiene un 1 si hay una arista entre el vértice en la posición I de la lista LV y el de la posición J de LV. En caso contrario tiene un 0. La matriz se representa como una lista de listas.

Solución:

```
matriz_adyacencia(LV, M) :-
    matriz_adyacencia_aux(LV, LV, M).

%matriz_adyacencia_aux(+LV, +LVPendientes, ?M) <- LVPendientes son las filas que quedan
por procesar para la matriz de adyacencia M, correspondientes a los vértices LV.
matriz_adyacencia_aux(_, [], []).

matriz_adyacencia_aux(LV, [VPendiente|RestoLVPendientes], [PrimeraFila|RestoFilas]) :-
    fila_adyacencia(LV, VPendiente, PrimeraFila),
    matriz_adyacencia_aux(LV, RestoLVPendientes, RestoFilas).

fila_adyacencia([], _, []).

%fila_adyacencia(+LV, +V, ?Fila) <- Fila es la fila correspondiente al vértice V en la
matriz de adyacencia de la lista de vértices LV.
fila_adyacencia([PrimeroLV|RestoLV], V, [1|RestoFila]) :-
    arista(V, PrimeroLV),
    fila_adyacencia(RestoLV, V, RestoFila).

fila_adyacencia([PrimeroLV|RestoLV], V, [0|RestoFila]) :-
    \+ arista(V, PrimeroLV),
    fila_adyacencia(RestoLV, V, RestoFila).
```

b) `traza(+M, ?T) <-` T es la cantidad de lazos del grafo (vértices que están relacionados con ellos mismos), que en este caso coincide con el valor de la traza de la matriz de adyacencia M (la traza de una matriz es la suma de los elementos de la diagonal). Este predicado no utiliza al predicado `arista` como auxiliar.

Solución:

```
traza(M, T) :-
    traza_ac(1, 0, M, T).

%traza_ac(+N, +Ac, +M, ?T) <- T es la traza de la matriz M desde la fila N. Ac es un
acumulador de T.
traza_ac(_, T, [], T).

traza_ac(N, Ac, [PrimeraFila|RestoFilas], T):-
    posicion(N, PrimeraFila, V),
    AcMasV is Ac + V,
    N1 is N + 1,
    traza_ac(N1, AcMasV, RestoFilas, T).

%posicion(+N, +L, ?V) <- V es elemento en la posición N de la lista L.
posicion(1, [Primero|_], Primero).

posicion(N, [_|Resto], V) :-
```

```
  N > 1,
  N1 is N - 1,
  posicion(N1, Resto, V).
```

c) `camino_simple(+V, +U, ?C)` <- C es un camino simple (que no repite vértices) del vértice V al vértice U del grafo representado en el programa mediante el predicado arista. C es representado como una lista de vértices, que comienza con V y termina con U. Nota: siempre existe el camino trivial [v] desde un vértice a sí mismo, independientemente de que el vértice tenga un lazo o no.

```
camino_simple(V, U, C) :-
    camino_simple_aux(V, U, [], C).

%camino_simple_aux(?V, ?U, +Visitados, ?C) <- C es un camino simple de V a U que no pasa
por los vértices de la lista Visitados.
camino_simple_aux(V, V, Visitados, [V]) :-
    \+ member(V, Visitados).

camino_simple_aux(V, U, Visitados, [V|RestoC]) :-
    \+ member(V, Visitados),
    arista(V, W),
    camino_simple_aux(W, U, [V|Visitados], RestoC).

member(X, [X|_]).

member(X, [_|Resto]):-
    member(X, Resto).
```

d) `clausura_transitiva(+V, -LV)` <- LV es la lista de vértices a los cuales se llega haciendo recorridas sobre el grafo representado en el programa mediante el predicado arista, desde el vértice V. La lista incluye a V y puede tener elementos repetidos.

Solución:

```
clausura_transitiva(V, LV) :-
    findall(U, camino_simple(V, U, _), LV).
```

Ejercicio 5 [12 puntos]

En Programación Lógica Inductiva (algoritmo FOIL) se genera una o más cláusulas para un predicado objetivo a partir de ejemplos positivos y negativos y cláusulas opcionales de background.

a) Explique brevemente qué significa que una cláusula es **consistente** respecto al conjunto de ejemplos y al conocimiento background.

La cláusula y el conocimiento background no implican ningún ejemplo negativo.

b) Explique brevemente qué significa que una cláusula es **completa** respecto al conjunto de ejemplos y al conocimiento background.

La cláusula y el conocimiento background implican todos los ejemplos positivos.

c) Considere el siguiente caso de ILP, con el predicado `camino(X,Y)` como predicado objetivo a aprender e hipótesis de mundo cerrado para la negación.

Background

```
arista(a,b), arista(c,d), arista(a,c), arista(b,e)
```

Ejemplos

```
camino(a,b)
camino(c,d)
camino(a,c)
camino(b,e)
camino(a,e)
camino(a,d)
```

Para cada una de las siguientes cláusulas indique si es consistente y/o completa. Justifique brevemente:

```
i. camino(X,Y) :- arista(X,Y).
```
Es consistente (no implica ejemplos negativos), no es completa (no cubre camino(a,e)).

```
ii. camino(X,Y).
```
Es inconsistente (implica camino(e,a), que es un ejemplo negativo), es completa ya que cubre todos los ejemplos positivos

```
iii. camino(X,Y) :- arista(X,Z),arista(Z,Y).
```
Es consistente y no es completa. Notar que aunque en el conjunto de ejemplos todos los caminos se hacen con a lo sumo 2 pasos de la relación arista, este predicado no se cumple con los caminos compuestos de una sola arista.