Evaluación (junio 2017)

Duración: 3 horas

Ejercicio 1 [25 puntos]

Se quiere resolver el problema de la asignación de invitados a mesas en una fiesta. Se conoce la lista de invitados y cuáles de ellos se conocen entre sí, y se los quiere distribuir en un conjunto de mesas que pueden acomodar diferentes cantidades de comensales. Se dice que una asignación a mesas es *válida* si se cumple que para cada mesa, todos los invitados de la mesa conocen por lo menos a otro invitado ubicado en la misma mesa. El problema se dividirá en dos partes:

a) Se cuenta con el siguiente predicado definido en la base de cláusulas:
`invitados(L)` <- **`L`** es la lista de invitados a la fiesta, por ejemplo: **`L = [claudia,pablo,ana,...]`**

Implementar en Prolog el predicado de asignación de invitados a mesas. Las asignaciones de invitados obtenidas mediante este predicado deben distribuir a todos los invitados en las mesas, pero pueden no ser válidas en cuanto a los invitados que se conocen.
`asignacion_mesas(+Mesas,?Asignacion)` <- **`Asignacion`** es una asignación de invitados a mesas, donde:

- **`Mesas`** es una lista de números naturales que indican la cantidad de invitados que puede acomodar cada mesa, por ejemplo: **`[4,3,4,2]`**. Puede asumir que la cantidad total de comensales que se pueden acomodar en las mesas es exactamente igual a la cantidad de invitados.
- **`Asignacion`** es una lista de listas, donde cada sublista indica una asignación de invitados a una mesa, por ejemplo: **`[[ana,juan,pedro,claudia],[pablo,gabriela,maria],...]`**

b) Se cuenta con el siguiente predicado definido en la base de cláusulas:
`conoce(X,Y)` <- El invitado **`X`** conoce al invitado **`Y`**

Implementar en Prolog el predicado de asignación válida de invitados a mesas:
`asignacion_valida(+Mesas,?Asignacion)` <- **`Asignacion`** es una asignación válida de invitados a mesas, o sea una asignación que cumple que todos los invitados de una mesa conocen al menos a otro invitado más de la misma mesa.

Ejercicio 2 [20 puntos]

Para cada uno de los siguientes enunciados, indique si es verdadero o falso. Fundamente.

a) Un conjunto de cláusulas de Horn es insatisfactible sii todas son cláusulas son insatisfactibles.

b) Un conjunto de cláusulas de Horn es insatisfactible sii al menos una de sus cláusulas es insatisfactible.

c) Un conjunto de cláusulas definidas es siempre satisfactible.

d) Sea P un programa definido, C una consulta con variables X y Z , σ_1 = { X / Y , Z / 2}, σ_2 = { X / 2 , Z / 2} , σ_3= { X / Y , Z / W} sustituciones.

i. Si σ_1 es una respuesta correcta, σ_2 es una respuesta correcta.
ii. Si σ_1 es una respuesta computada, σ_2 es una respuesta computada.
iii. Si σ_2 es una respuesta computada, σ_2 es una respuesta correcta.
iv. Si σ_2 es una respuesta correcta, σ_3 es una respuesta correcta.

Ejercicio 3 [20 puntos]

Dado el siguiente conjunto de cláusulas:

```
a(X,X) :- b(X), c(X).
a(X,Z) :- Y is X+1, a(Y,Z).
b(X) :- X > 1.
b(X) :- d(X).
c(0).
c(2).
d(0).
d(1).
```

a) Dibuje el árbol SLD correspondiente a la consulta **`?- a(0,W)`**, suponiendo que la regla de computación toma el átomo de más a la izquierda para la resolución.

b) ¿Qué respuestas dará un intérprete Prolog que aplique la regla de computación de la parte a), tomando las cláusulas en su orden de aparición y haciendo recorrida en profundidad del árbol?

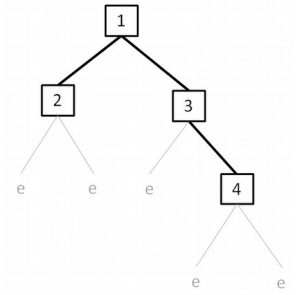
c) Diga cuáles serían las respuestas de la parte b) si las cláusulas correspondientes al predicado **a** estuvieran en orden inverso.

d) Indique qué ramas del árbol de la parte a) son podadas por los cuts de las siguientes variantes del conjunto de cláusulas de **a**:

i.
```
a(X,X) :- b(X), c(X), !.
a(X,Z) :- Y is X+1, a(Y,Z).
```

ii.
```
a(X,X) :- b(X), c(X).
a(X,Z) :- Y is X+1, !, a(Y,Z).
```

Ejercicio 4 [10 puntos]



Considere árboles binarios de valores enteros implementados mediante términos Prolog de la siguiente manera:
- El átomo **e** representa un árbol vacío.
- El término **t(Num,Izq,Der)** representa un árbol binario con raíz **Num** (un número entero), un subárbol izquierdo **Izq** y un subárbol derecho **Der**.

Por ejemplo, el término **t(1,t(2,e,e),t(3,e,t(4,e,e)))** representa el árbol de la derecha.

Implementar el siguiente predicado:
pos_orden(+T, ?L) <- **L** es una recorrida en pos orden de los elementos numéricos del árbol **T**. El predicado a implementar debe funcionar de forma que la lista a construir se recorra una sola vez.
(Sugerencia: utilizar listas de diferencias)

Ejemplo: pos_orden(t(1,t(2,e,e),t(3,e,t(4,e,e))),[2,4,3,1])

Ejercicio 5 [10 puntos]

Sean los siguientes conjuntos de hechos E+ y E-.

E+ = { P(0), P(s^3(0)), P(s^6(0)) }

E- = { P(s^2(0)), P(s^4(0)) }

En ILP buscamos definir una teoría (conjunto de cláusulas) que cubra todos los ejemplos positivos (E+) y no cubra ningún ejemplo negativo. Si la teoría cubre algún ejemplo negativo, diremos que es inconsistente; si cubre todos los ejemplos positivos diremos que es completa.

a. Se define Ê- = {e / ┐e ϵ E-}. Indique que condición se debe cumplir entre una teoría ∑ y Ê- para que ∑ no sea inconsistente.

b. Sean ∑1 = { P(s^2(x)) <- P(x) } , ∑2 = { P(s^3(x)) <- P(x) , P(s(0)) } dos teorías sobre el mismo alfabeto que los conjuntos E+ y E-

Indicar, para cada una de las teorías ∑1 y ∑2, si son correctas. Fundamentar.