Práctico 6: Estructuras Incompletas. Cut. Negación.

Ejercicio 1

Escriba los siguientes predicados para transformar listas en listas de diferencias y listas de diferencias en listas.

l_ld(*L*, *LD*) ← *LD* es una lista de diferencias equivalente a la lista *L*.

ld_l(*LD*, *L*) ← *L* es la lista equivalente a la lista de diferencias *LD*.

Ejercicio 2

Escriba los siguientes predicados para recorridas de árboles usando listas de diferencias:

pre_orden(*A*, *L*) ← *L* es una lista con los elementos del árbol binario *A*, obtenida al recorrerlo pre-orden.

in_orden(*A*, *L*) ← *L* es una lista con los elementos del árbol binario *A*, obtenida al recorrerlo in-orden.

post_orden(*A*, *L*) ← *L* es una lista con los elementos del árbol binario *A*, obtenida al recorrerlo post-orden.

Ejercicio 3 [prueba 02]

Sea el siguiente predicado *strange*:

```
strange(Xs, Ys, P)             :- strange(Xs, Ys, [], P).
strange([X|Xs], [X|Ys], Zs, P) :- X<P, !, strange(Xs, Ys, Zs, P).
strange([X|Xs], Ys, Zs, P)     :- strange(Xs, Ys, [X|Zs], P).
strange([], Xs, Xs, _).
```

a) Indique las respuestas a las siguientes consultas:

i. `strange([1,5,8,9], Ys, 7).`
ii. `strange([1,5,8,9], [1,5,9,8], 8).`
iii. `strange([1,5,8,9], [1,5,8,9], 8).`
iv. `strange(Xs, [1,2,3,4], 4).`

b) Explique brevemente qué hace el predicado *strange*, e indique qué instanciación de argumentos admite.

c) A su entender, ¿es el cut del predicado *strange* verde o rojo? Justifique.

Ejercicio 4 [prueba 06]

Sea el siguiente programa *Prolog*:

```
arbol(arb(prolog,
            arb(3,
                  h(fing),
                  h(4)
            ),
            arb(fing,
                  arb(prolog,
                        h(1),
                        h(2)
                  ),
                  h(inco)
            )
      )
).
```

```
achatable(fing).
achatable(prolog).
achata(h(X),[X|Xs], Xs).
achata(arb(X,Y,Z),[X|Ys],Rs):-
                        achatable(X),
                        achata(Y,Ys,Zs),
                        achata(Z,Zs,Rs).
achata(arb(X,Y,Z), Ys, Rs)  :-
                        achata(Y,Ys,Rs).
```

y la consulta: *?arbol(Arbol), achata(Arbol , As, []).*

a) Dé los valores de *As* que son solución de la consulta. Justifique.
b) Cree variantes del programa anterior, de forma tal que, agregando uno y solamente un *cut* a las cláusulas de achata/3, la consulta tenga por resultado:
   i. una única solución
   ii. ninguna solución
   iii. las mismas soluciones
c) Si se agrega un cut a la primera cláusula de achatable/1, ¿se modifica la respuesta a la consulta anterior? En caso afirmativo, muestre cuál es. En caso negativo, dé una consulta para la cual el resultado cambia.

Ejercicio 5

Considere el siguiente programa lógico P:

```
p(X)    :-  q(Y), r(Y).
q(h(Y)) :-  q(Y).
r(g(Y)).
```

Encuentre dos árboles SLD para P ∪ {←p(a)}, uno de los cuales sea infinito y el otro sea finitamente fallado.

Ejercicio 6 [prueba 00]

Sea el siguiente programa Prolog:

```
par(X)                 :- 0 is (X mod 2).
multiplo_tres(X)       :- 0 is (X mod 3).
sel_par([X|Xs], X, Z)  :- par(X), X > Z.
sel_par([X|Xs], Y, Z)  :- sel_par(Xs, Y, Z).
prob(X)                :- sel_par([1,4,12,5,32,30,17,18], X, 10),
                          multiplo_tres(X).
```

a) Dé los valores de *X* que son solución de la consulta *?- prob(X)*.
b) Para las siguientes variantes con *cut* del programa anterior, indique los valores de *X* que son solución de la consulta *?- prob(X)*. Justifique sus respuestas.

i.

```
par(X)                 :- 0 is (X mod 2).
multiplo_tres(X)       :- 0 is (X mod 3).
sel_par([X|Xs], X, Z)  :- par(X), X > Z, !.
sel_par([X|Xs], Y, Z)  :- sel_par(Xs, Y, Z).
prob(X)                :- sel_par([1,4,12,5,32,30,17,18], X, 10),
                          multiplo_tres(X).
```

ii.

```
par(X)                 :- 0 is (X mod 2).

multiplo_tres(X)       :- 0 is (X mod 3).

sel_par([X|Xs], X, Z)  :- par(X), X > Z.

sel_par([X|Xs], Y, Z)  :- sel_par(Xs, Y, Z).

prob(X)                :- sel_par([1,4,12,5,32,30,17,18], X, 10),
                          !, multiplo_tres(X).
```

iii.

```
par(X)                 :- 0 is (X mod 2).

multiplo_tres(X)       :- 0 is (X mod 3).

sel_par([X|Xs], X, Z)  :- par(X), X > Z.

sel_par([X|Xs], Y, Z)  :- sel_par(Xs, Y, Z).

prob(X)                :- sel_par([1,4,12,5,32,30,17,18], X, 10),
                          multiplo_tres(X), !.
```

iv.

```
par(X)                 :- 0 is (X mod 2).

multiplo_tres(X)       :- 0 is (X mod 3).

sel_par([X|Xs], X, Z)  :- par(X), X > Z.

sel_par([X|Xs], Y, Z)  :- sel_par(Xs, Y, Z), !.

prob(X)                :- sel_par([1,4,12,5,32,30,17,18], X, 10),
                          multiplo_tres(X).
```

v.

```
par(X)                 :- 0 is (X mod 2).

multiplo_tres(X)       :- 0 is (X mod 3).

sel_par([X|Xs], X, Z)  :- par(X),!, X > Z.

sel_par([X|Xs], Y, Z)  :- sel_par(Xs, Y, Z).

prob(X)                :- sel_par([1,4,12,5,32,30,17,18], X, 10),
                          multiplo_tres(X).
```

vi.

```
par(X)                 :- 0 is (X mod 2), !.

multiplo_tres(X)       :- 0 is (X mod 3).

sel_par([X|Xs], X, Z)  :- par(X), X > Z.

sel_par([X|Xs], Y, Z)  :- sel_par(Xs, Y, Z).

prob(X)                :- sel_par([1,4,12,5,32,30,17,18], X, 10),
                          multiplo_tres(X).
```

Ejercicio 7
Defina los siguientes metapredicados utilizando, de ser necesario, **cut** y **fail**:

or(Goal1, Goal2)	←	Se satisface si alguno de los argumentos lo hace.
not(Goal)	←	Se satisface si *Goal* tiene un árbol SLD finitamente fallado (según la regla de computación de Prolog).
once(Goal)	←	Se satisface una **única** vez, si *Goal* se satisface al menos una vez.
ignore(Goal)	←	Análogo al predicado *once*, pero se satisface siempre, aunque *Goal* no lo haga.
if_then(If, Then)	←	Se satisface si *If* y *Then* se satisfacen.
if_then_else(If, Then, Else)	←	Se satisface si *If* y *Then* se satisfacen, o si no se satisface *If*, pero sí lo hace *Else*.

Notar que todos los argumentos son metavariables (objetivos). Sólo se debe considerar una única forma de satisfacer el objetivo *If*.

Ejercicio 8
Sea G un grafo no orientado con costos positivos, representado por las relaciones: *arista*(G, V1, V2, Costo) y *vertice*(G, V1).

Defina predicados Prolog utilizando **not** para:

conexo(G)	←	El grafo *G* es conexo (existe un camino entre todo par de nodos).
completo(G)	←	Existe una arista entre todo par de vértices

Ejercicio 9
Considere el siguiente programa:

```
p  :- ( a, !
      ; c, (d, !, e), f
      ),
      call((g,!)),
      not(h, !, i).
p  :- ...
```

¿Cuáles cuts de la primera cláusula para *p* cortan la segunda alternativa? ¿Puede dar una regla para determinar el «alcance» de un cut?