Práctico 7: Cut y Negación.

Ejercicio 1

a) Considere que C1, C2 y C3 son conjuntos representados como listas sin elementos repetidos. Implemente los siguientes predicados en Prolog puro más el predicado \= para chequear que dos elementos sean diferentes.

i. `intersec(+C1,+C2,?C3)` ← C3 es la intersección de los conjuntos C1 y C2.
ii. `diferencia(+C1,+C2,?C3)` ← C3 es el conjunto C1 - C2.

b) Mejore la eficiencia de los predicados de la parte a) utilizando **cut** de manera que no se recorra la segunda lista lista innecesariamente.

Ejercicio 2 [prueba 16]

Sea el siguiente programa lógico:

```
p(X,Y,Z):-                  q(X,X,[X]):-
    q(X,Y,Z).                   X > 0.
p(X,Y,Z):-                  q(X,Y,[X|Z]):-
    N is X + 1,                 N is X - 1,
    p(N,Y,Z).                   q(N,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.
b) Indique qué respuestas dará el intérprete de Prolog para el objetivo anterior.
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, !.
```

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, _).
```

d) 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).`

e) Explique brevemente qué hace el predicado *strange*, e indique qué instanciación de argumentos admite.
f) 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 [prueba 00]

Sea el siguiente programa Prolog:

```
[1] par(X) :- 0 is (X mod 2).
[2] multiplo_tres(X) :- 0 is (X mod 3).
[3] sel_par([X|Xs], X, Z) :- par(X), X > Z.
[4] sel_par([X|Xs], Y, Z) :- sel_par(Xs, Y, Z).
[5] 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. `[3] sel_par([X|Xs], X, Z) :- par(X), X > Z, !.`

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

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

iv. `[4] sel_par([X|Xs], Y, Z) :- sel_par(Xs, Y, Z), !.`

v. `[3] sel_par([X|Xs], X, Z) :- par(X), !, X > Z.`

vi. `[1] par(X) :- 0 is (X mod 2), !.`

Ejercicio 6

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

or(Goal1, Goal2)	←	Se satisface si alguno de los argumentos lo hace.
if_then(If, Then)	←	Si *If* se satisface *Then* se debe satisfacer.
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*.
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.

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

Ejercicio 7

Considere que C1, C2 y C3 son conjuntos representados como listas sin elementos repetidos. Implemente los siguientes predicados en Prolog utilizando **not**:

i. `diferencia(+C1,+C2,?C3)` ← C3 es el conjunto C1 - C2.
ii. `disjuntos(+C1,+C2)` ← C1 y C2 son disjuntos.

Ejercicio 8

Sea G un grafo dirigido representado por las relaciones: *arista*(G, V1, V2) y *vertice*(G, V1).

Defina predicados Prolog utilizando **not** para:

completo(G)	←	Existe una arista entre todo par de vértices
hay_camino(G,A,B)	←	Existe un camino entre A y B para el grafo G. *Cuidado con los ciclos del grafo!*
conexo(G)	←	El grafo *G* es conexo (existe un camino 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?