Programación Declarativa

3^{er} Curso, Grado en Ingeniería en Informática
Universidad Rey Juan Carlos

Programación Lógica

Compendio de exámenes resueltos

Este documento recopila los exámenes de Programación Lógica, resueltos, correspondientes a los últimos cursos, tanto de la convocatoria ordinaria como de la extraordinaria.

Prueba Programación Lógica

29-6-2022

Apellidos: ______________________ **Nombre:** ______________

Grado: ______________________

- La duración de esta prueba es de 1 hora y 20 minutos.
- No se permite el uso de ningún tipo de material ni dispositivo electrónico.

Además de los predicados aritméticos, el corte y la negación, se podrán utilizar los siguientes: `length(?L,?N)` (cierto si `N` es la longitud de la lista `L`), `append(?L1,?L2,?L)` (cierto si `L` es la concatenación de las listas `L1` y `L2`), `member(?E,?L)` (cierto si `E` pertenece a `L`), `is_list(+L)` (cierto si `L` es una lista), `sumlist(+L,?N)` (cierto si `L` es una lista de números cuya suma es `N`), `maplist(+Obj,+L)` (cierto si `Obj` se ejecuta con éxito sobre todos los elementos de la lista `L`), `maplist(+Obj,?L1,?L2)` (cierto si `Obj` se ejecuta con éxito sobre todas las parejas de elementos de las listas `L1` y `L2` situados en la misma posición), `include/exclude(+Obj,+L1,?L2)` (cierto si `L2` incluye/excluye los elementos de la lista `L1` sobre los que se ejecuta con éxito `Obj`), `pliegai(+Obj,+L,+VI,-VF)`, cierto si `VF` es el resultado de plegar mediante `Obj` la lista `L` desde la izquierda partiendo de `VI`), `number/integer/var/nonvar(+N)` (cierto si `N` es un número/entero/variable/novariable), `call(+Obj,+E1,..)` (cierto si `Obj`, con parámetros extra `E1,..`, se ejecuta con éxito), y los predicados de recolección `bagof/setof/findall(?T, +Obj, ?L)`.

1. (2.5 puntos) Suponga disponibles los predicados `habitantes(?P,?N)`, cierto si el país `P` tiene `N` millones de habitantes, `frontera(?P1,?P2)`, cierto si `P1` y `P2` hacen frontera, y `enemigos(+P,?L)`, cierto si `L` es la lista de los países enemigos de `P`. Implemente el predicado `hab(+P,?L,?N)`, cierto si `L` es una lista *ordenada* de los países fronterizos con `P` que no son enemigos suyos y `N` es la suma (en millones) de los habitantes de `P` con los de sus vecinos no enemigos. Si `P` no tuviese vecinos o todos fuesen enemigos, `hab(P,L,N)` deberá ser cierto con `L=[]` y `N` igual a los habitantes de `P`. Tenga en cuenta que en la base de conocimientos del programa, el predicado `frontera` se utiliza presuponiendo su simetría, de forma que para indicar que `a` y `b` son fronterizos basta con el hecho `frontera(a,b)` o bien el hecho `frontera(b,a)` (indistintamente).

Solución:

```
hab(P, L, N) :-
    enemigos(P,EP),
    setof(V,
          ((frontera(P,V) ; frontera(V,P)), \+ member(V,EP)),
          L),
    !,
    maplist(habitantes, [P|L], LHabs),
    sumlist(LHabs, N).
```

```
hab(P, [], N) :-
    habitantes(P, N).
```

2. (2 puntos) Dado el predicado `que(+N, +L, ?NL)` implementado como sigue:

```
que(N, L, NL) :-
    findall(X,
            (append(A, [B], L), member(X, A), N > X),
            NL).
```

(a) (0.5 puntos) **Razone** qué se obtendría (un error, true, false, computación infinita, respuesta(s) que sería(n) ...) al ejecutar en Prolog la consulta

```
?- que(2, [5,4,3,2,1], U).
```

Solución:
`[B]` solo unifica con listas de un elemento y por lo tanto solo puede ser `B=1` y `A = [5,4,3,2]`. El predicado member va recorriendo, por backtraking, la lista anterior, y evalúa para cada X la condición `2>X`. Como esta condición resulta falsa para todos los X de A, el predicado findall no recolecta ningún X y por lo tanto `que` termina con `U=[]`.

(b) (1 punto) Proponga una implementación alternativa para el predicado "`que`" en la que *no* se utilice ningún predicado de recolección. Se valorará su concisión.

Solución:

```
que(_, [], []).
que(N, L, NL) :-
   append(A, [_], L),
   include(>(N), A, NL).
```

(c) (0.5 puntos) Misma pregunta que en el apartado (a) pero intercambiando de sitio los dos primeros argumentos de `append` en el código de `que` (e.d. escribiendo `append([B], A, L)`).

Solución:
Con el cambio introducido el predicado `append` solo puede ser cierto con `B=5` y `A = [4,3,2,1]`. El predicado member va recorriendo, por backtraking, la lista anterior, y evalúa para cada X la condición `2>X`. Lo anterior solo es cierto con X=1, por lo que la consulta devuelve cierto con `U=[1]`.

3. (3.5 puntos) Considere el predicado `dw(+L, +Obj, ?NL)`, cierto si `NL` es el sufijo de `L` que empieza con el primer elemento (contando desde la izquierda) que *no* cumple `Obj`. `NL` será la propia `L` si el primer elemento de esta no cumple `Obj`, y vacía si `L` lo es o todos sus elementos cumplen `Obj`. Algunos ejemplos:

   - `dw([1,2,a,3], integer, NL)` daría `NL = [a,3]`.
   - `dw([a,1,2], integer, NL)` daría `NL = [a,1,2]`.
   - `dw([], integer, NL)` o `dw([1,2,3], integer, NL)` darían `NL = []`.

   (a) (2 puntos) Proponga una implementación **recursiva** para el predicado `dw`.

   Solución:

```
dw([C|R], Obj, NR) :-
    call(Obj, C),
    !,
    dw(R, Obj, NR).

dw(L, _, L).
```

   (b) (1.5 puntos) Proponga una implementación **no recursiva** para el predicado `dw`, basada en el uso de alguno(s) de los predicados mencionados al comienzo del enunciado.

   Solución:

```
dwa(L, Obj, NL) :-
     append(_, [C|R], L),
     \+ call(Obj, C),
        !,
     NL = [C|R].

dwa(_, _, []).
```

4. (2 puntos) Dados el programa y la consulta que se incluyen en la otra cara de esta hoja, **dibuje** el Árbol de Resolución correspondiente, etiquetando cada arco con su unificador de máxima generalidad y marcando claramente las posibles ramas podadas, ramas fallo, ramas infinitas o ramas éxito. En caso de haber soluciones, estas solo se valorarán si se acompañan de los cálculos que permiten obtenerlas. Indique además **qué respuesta(s)** ofrecería Prolog ante la consulta dada, y en qué orden lo haría.

```
s(L, L).
s(L, [_|R]) :- s(L, R).

              ?- s([C|D], [1,a]), \+ (D = []).
```

Solución:

El árbol de Resolución correspondiente (en el examen había que dibujarlo) tiene, recorriéndolo en profundidad de izquierda a derecha, las siguientes ramas: una rama fallo, una rama éxito con solución `C=1,D=[a]` una rama fallo, una rama podada y una rama fallo.

Además de dibujar el árbol, de acuerdo con el enunciado, era necesario:

- Acompañar cada una de las soluciones de los cálculos que permiten obtenerlas (aplicando a cada una de las variables de la consulta los umg's de la rama, en orden, empezando por la raíz).
- Indicar qué respuestas ofrece Prolog ante la consulta dada y en qué orden las da: dado que Prolog construye el árbol en profundidad por la izquierda, ignorando tanto ramas fallo como ramas podadas, la única respuesta sería `C=1,D=[a]` (y a continuación `false` indicando que no hay más soluciones).

Universidad Rey Juan Carlos

Prueba de Programación Lógica

21-1-2022

Apellidos: ______________________ **Nombre:** ______________

Grado: ______________________

- La duración de esta prueba es de 1 hora y 30 minutos.
- No se permite el uso de ningún tipo de material ni dispositivo electrónico.

Además de los predicados aritméticos, el corte y la negación, se podrán utilizar los siguientes: `length(?L,?N)` (cierto si `N` es la longitud de la lista `L`), `append(?L1,?L2,?L)` (cierto si `L` es la concatenación de las listas `L1` y `L2`), `member(?E,?L)` (cierto si `E` pertenece a `L`), `is_list(+L)` (cierto si `L` es una lista), `sumlist(+L,?N)` (cierto si `L` es una lista de números cuya suma es `N`), `maplist(+Obj,+L)` (cierto si `Obj` se ejecuta con éxito sobre todos los elementos de la lista `L`), `maplist(+Obj,?L1,?L2)` (cierto si `Obj` se ejecuta con éxito sobre todas las parejas de elementos de las listas `L1` y `L2` situados en la misma posición), `include/exclude(+Obj,+L1,?L2)` (cierto si `L2` incluye/excluye los elementos de la lista `L1` sobre los que se ejecuta con éxito `Obj`), `pliegai(+Obj,+L,+VI,-VF)`, cierto si `VF` es el resultado de plegar mediante `Obj` la lista `L` desde la izquierda partiendo de `VI`), `number/integer/var/nonvar(+N)` (cierto si `N` es un número/entero/variable/novariable), `call(+Obj,+E1,..)` (cierto si `Obj`, con parámetros extra `E1,..`, se ejecuta con éxito), y los predicados de recolección `bagof/setof/findall(?T, +Obj, ?L)`.

1. (1 punto) Dado el predicado `que(+L, ?NL)` implementado como sigue

```
que(L, NL) :-
    bagof(X,
          P^Q^LP^(append(P, [X|Q], L), length(P, LP), LP mod 2 =:= 1),
          NL).
```

(a) (0.8 puntos) Describa en lenguaje natural (con sus propias palabras) el *resultado* de `que(L,NL)` (`que(L,NL)` es cierto si ...).

Solución:
`que(L,NL)` es cierto si `NL` es la lista conteniendo los elementos de la lista `L` situados en posiciones pares (entendiendo que la cabeza de `L` está en la posición 1).

(b) (0.2 puntos) Suponga que `X` es una lista e `Y` una variable. ¿Podría la consulta `?- que(X,Y)` devolver `false`? En caso afirmativo, ¿en qué circunstancias?

Solución:
El predicado `bagof` falla cuando no es capaz de recolectar ningún término cumpliendo los objetivos dados. Por ello, con las suposiciones dadas, el predicado `que(X,Y)` fallará si y solo si la lista `X` no tiene ningún elemento situado en posiciones pares, es decir, cuando tenga tamaño menor o igual que 1.

2. (4 puntos) Considere el predicado `fm(+Obj1, +Obj2, +L, ?NL, ?N)` dado por

```
fm(Obj1, Obj2, L, NL, N) :-
    include(Obj1, L, L1),
    maplist(Obj2, L1, NL),
    length(NL, N).
```

(a) (1 punto) Proponga tres consultas concretas con `fm` tales que una produzca un error, otra devuelva `false` (**razone** en ambos casos por qué) y la tercera, con los dos últimos parámetros de salida, devuelva una solución (indique cuál sería).

Solución:
Siendo `cuadrado(X,Y) :- Y is X^2.`:

- La consulta `?- fm(>, cuadrado, [2,a,X,3], L, Y).` produce un error (debido a que el predicado `include` intenta ejecutar `>(2)`, cuando `>` es una relación con dos parámetros, no con uno).
- La consulta `?- fm(integer, cuadrado, [2], [3], N).` devuelve false (debido a que el predicado `maplist` ejecuta `cuadrado(2,3)`, que falla).
- La consulta `?- fm(integer, cuadrado, [2,a,X,3], L, N).` devuelve `L = [4,9], N = 2`.

(b) (3 puntos) Facilite una implementación ***recursiva*** para `fm`. Se valorará que presente *recursión de cola (recursión final)*.

Solución:

```
%% Implementación recursiva (con recursión no final)

% caso base
fm_r(_Obj1, _Obj2, [], [], 0).

% caso recursivo en el que C cumple Obj1
fm_r(Obj1, Obj2, [C|R], L, N) :-
    call(Obj1, C),
    !,
    call(Obj2, C, NC),
    fm_r(Obj1, Obj2, R, NR, NumR),
    L = [NC|NR],
    N is NumR + 1.

% caso recursivo en el que C NO cumple Obj1
fm_r(Obj1, Obj2, [_|R], NR, N) :-
```

```
    fm_r(Obj1, Obj2, R, NR, N).


%% Implementación con recursión final (recursión de cola)

fm_rc(Obj1, Obj2, L, NL, N) :-
   % acumulador en el penúltimo parámetro
    fm_rc(Obj1, Obj2, L, NL, 0, N).

% caso base: se devuelve lo acumulado
fm_rc(_Obj1, _Obj2, [], [], Acc, Acc).

% caso recursivo en el que C cumple Obj1: se acumula 1 más
fm_rc(Obj1, Obj2, [C|R], L, Acc, N) :-
    call(Obj1, C),
    !,
    call(Obj2, C, NC),
    NAcc is Acc + 1,
    L = [NC|NR],
    fm_rc(Obj1, Obj2, R, NR, NAcc, N).

% caso recursivo en el que C NO cumple Obj1: no se acumula
fm_rc(Obj1, Obj2, [_|R], NR, Acc, N) :-
    fm_rc(Obj1, Obj2, R, NR, Acc, N).
```

3. (1 punto) Suponga que en el código `fm` del ejercicio anterior las dos primeras líneas del cuerpo de la regla se sustituyen por "`maplist(Obj2,L,L1),include(Obj1,L1,NL)`". **Razone** si esta modificación podría alterar o no el resultado del predicado `fm`.

Solución:

La modificación puede alterar el resultado del predicado puesto que, en general, no es lo mismo filtrar primero y luego transformar que lo contrario. Por ejemplo, la primera consulta propuesta en el primer apartado del ejercicio anterior seguiría dando un error pero por otro motivo (el error se produce al intentar calcular el cuadrado de `a`, el include no llega a ejecutarse) y la tercera ahora daría un error (al intentar calcular el cuadrado de `a`). Otro ejemplo: `fm(>(4), cuadrado, [2,3], L, N)` daría `L = [4,9], N = 2` con la implementación original y `L = [], N = 0` después del cambio.

4. (2 puntos) Considere disponibles los predicados `ancestro(X,Y)`, cierto si `X` es un ancestro (progenitor, abuelo, bisabuelo, etc) de `Y`, `edad(X,E)`, cierto si `X` tiene `E` años y `en_paro(L)`, cierto si `L` es una lista con las personas que están en paro. Proponga una implementación para `ex(+P, ?N, ?Me, ?Ma)`, cierto si `P` tiene `N>0` ascendientes mayores de edad que no están en paro y `Me/Ma` son, respectivamente, las edades

del menor/mayor de ellos. Por ejemplo, si se tuviese la familia `p1-p2-p3-(p31,p32)` donde cada uno es progenitor del o de los siguientes, con edades `80-60-40-(15,10)`y donde el único que está en paro es `p2`, la consulta `ex(p32,N,Me,Ma)` daría cierto con `N=2,Me=40,Ma=80` (lo mismo para `p31`), `ex(p3,N,Me,Ma)` daría cierto con `N=1,Me=80,Ma=80` (lo mismo para `p2`) y `ex(p1,N,Me,Ma)` fallaría.

Solución:

```
ex(P, N, Men, May) :-
     en_paro(Pds),
     bagof(E,       % puede haber edades repetidas
           A^(ancestro(A,P), \+ member(A,Pds), edad(A,E), E >= 18),
           LEs),
     length(LEs, N),
     setof(X, member(X, LEs), LEsOrd), % ordena de menor a mayor
     [Men|_] = LEsOrd,                 % el primero es el menor
     append(_, [May], LEsOrd).         % el último es el mayor
```

5. (2 puntos) Dados el programa y la consulta que se incluyen en la otra cara de esta hoja, **dibuje** el Árbol de Resolución correspondiente, etiquetando cada arco con su unificador de máxima generalidad y marcando claramente las posibles ramas podadas, ramas fallo, ramas infinitas o ramas éxito. En caso de haber soluciones, estas solo se valorarán si se acompañan de los cálculos que permiten obtenerlas. Indique además **qué respuesta(s)** ofrecería Prolog ante la consulta dada, y en qué orden lo haría.

```
m(C, [ C | _ ]).
m(C, [ _ , _ | R]) :- m(C, R).

?- m(X,[[1,3,2],[3,4],[5,6]]), !, m(Y,X), \+ (Y mod 2 =:= 0), Z is Y+1.
```

Solución:

El árbol de Resolución correspondiente (en el examen había que dibujarlo) tiene, recorriéndolo en profundidad de izquierda a derecha, las siguientes ramas: rama fallo, rama éxito con solución `X=[1,3,2],Y=1,Z=2`, rama fallo y dos ramas podadas por cortes.

Además de dibujar el árbol, de acuerdo con el enunciado, era necesario:

- Acompañar cada una de las soluciones de los cálculos que permiten obtenerlas (aplicando a cada una de las variables de la consulta los umg's de la rama, en orden, empezando por la raíz).
- Indicar qué respuestas ofrece Prolog ante la consulta dada y en qué orden las da: dado que Prolog construye el árbol en profundidad por la izquierda,

ignorando tanto ramas fallo como ramas podadas, la única respuesta sería `X=[1,2,2],Y=1,Z=2` y a continuación `false` (indicando que no hay más soluciones).

Universidad Rey Juan Carlos

Prueba de Programación Lógica A

2-7-2021

Apellidos: ______________________ **Nombre:** ______________

Grado: ______________________

- La duración de esta prueba es de 1 hora y 20 minutos.
- No se permite el uso de ningún tipo de material ni dispositivo electrónico.

Además de los predicados aritméticos, el corte y la negación, se podrán utilizar los siguientes: `length(?L,?N)` (cierto si `N` es la longitud de la lista `L`), `append(?L1,?L2,?L)` (cierto si `L` es la concatenación de las listas `L1` y `L2`), `reverse(?L1,?L2)` (cierto si `L2` es la inversa de la lista `L1`), `member(?E,?L)` (cierto si `E` pertenece a `L`), `is_list(+L)` (cierto si `L` es una lista), `sumlist(+L,?N)` (cierto si `L` es una lista de números cuya suma es `N`), `map(+Obj,+L)` (cierto si `Obj` se ejecuta con éxito sobre todos los elementos de la lista `L`), `map(+Obj,?L1,?L2)` (cierto si `Obj` se ejecuta con éxito sobre todas las parejas de elementos de las listas `L1` y `L2` situados en la misma posición), `include/exclude(+Obj,+L1,?L2)` (cierto si `L2` incluye/excluye los elementos de la lista `L1` sobre los que se ejecuta con éxito `Obj`), `pliegai(+Obj,+L,+VI,-VF)`, cierto si `VF` es el resultado de plegar mediante `Obj` la lista `L` desde la izquierda partiendo de `VI`), `number/integer/var/nonvar(+N)` (cierto si `N` es un número/entero/variable/novariable), `call(+Obj,+E1,..)` (cierto si `Obj`, con parámetros extra `E1,..`, se ejecuta con éxito) y los predicados de recolección `bagof/setof/findall(?T, +Obj, ?L)`.

1. (1.5 puntos) **Razone** qué se obtendría (un error, true, false, computación infinita, respuesta(s) que sería(n) ...) al ejecutar en Prolog la siguiente consulta solicitando todas las posibles soluciones:

```
?- append([C|R], S, [4, 2, 3.2, 1]), map(integer, [C|R]),
   sumlist([C|R], N), N =< 10.
```

Solución:

El primer objetivo, el predicado `append`, descompone la lista `[4, 2, 3.2, 1]` en dos sublistas donde la primera, `[C|R]`, no puede ser vacía:

- La primera solución del predicado `append` es `[C|R]=[4]`, lo cual implica `C=4, R=[]`, y `S=[2, 3.2, 1]`. Con esos valores el `map` posterior es cierto (4 es un número entero), el `sumlist` subsiguiente unifica `N` con 4 y la última comprobación, `4 =< 10`, es cierta. Se trata por lo tanto de una rama éxito con solución:

  `C = 4, R = [], S = [2,3.2,1], N = 4`

- Al hacer backtracking, el único predicado con posibilidades de reevaluación es `append`, que encuentra ahora la descomposición `[C|R]=[4,2]`, lo cual implica `C=4, R=[2]`, y `S=[3.2, 1]`. Con esos valores el `map` posterior es cierto (4 y 2 son números enteros), el `sumlist` subsiguiente unifica `N` con 4+2=6 y la última comprobación, `6 =< 10`, es cierta. Se trata por lo tanto de una rama éxito con solución:

```
C = 4, R = [2], S = [3.2,1], N = 6
```

- Al hacer backtracking de nuevo, el predicado `append` proporciona otras dos soluciones, pero en ambas soluciones `R` contiene el elemento 3,2, número no entero, por lo que el `map` falla, haciendo fallar ambos intentos y acabando la consulta por lo tanto con un `false` que indica que no hay más soluciones.

2. (1 punto) Implemente el predicado `desde(+Min, +Max, ?L)`, cierto si `Min` y `Max` son dos números naturales tales que `Min` $\leq$ `Max` y `L` es la lista conteniendo todos los naturales comprendidos entre ellos (ambos incluidos) ordenados de menor a mayor. El predicado no debe producir ningún error en tiempo de ejecución. Por ejemplo, las consultas `desde(hola,2,L)` y `desde(2,1,L)` deben fallar, mientras que `desde(1,3,X)` y `desde(0,0,Y)` serían ciertas con `X=[1,2,3]` e `Y=[0]`.

Solución:

```
desde(Min, Max, L) :-
  integer(Min),
  integer(Max),
  Min >= 0,
  Max >= Min,
  desde_aux(Min, Max, L).

desde_aux(N, N, [N]) :- !.

desde_aux(Min, Max, [Min|R]) :-
  Min1 is Min + 1,
  desde_aux(Min1, Max, R).
```

3. (3.5 puntos) Considere el predicado `sumap(+N,?S)`, cierto si `N` es un número natural y `S` es la suma de todos los naturales pares menores o iguales que `N`. El predicado no debe producir ningún error en tiempo de ejecución. Por ejemplo las consultas `?-sumap(-2.3,S)` y `?-sumap(hola,2)` deben devolver `false`, mientras que `?-sumap(4,S)` y `?-sumap(5,S)` darían ambas `S=6` como única solución (4+2+0=6).

(a) (1 punto) Proponga una implementación *no recursiva* para `sumap` sabiendo que dispone del predicado `desde` del ejercicio anterior y de los predicados mencionados al comienzo del enunciado.

Solución:

```
sumap(N, S) :-
    desde(0,N,L),
    include(par, L, LP),
    sumlist(LP, S).

par(N) :-
   N mod 2 =:= 0.
```

(b) (2.5 puntos) Proponga una implementación para `sumap` que sea *recursiva* y no use (ni implemente) el predicado `desde`. Se valorará que la implementación presente *recursión de cola (recursión final).*

Solución:

```
%% Implementación recursiva (con recursión no final)

 sumap(N, S) :-
    integer(N),
    N >= 0,
    suma_aux(2, N, S).

 % suma_aux(Desde, Hasta, S)
 % cierto si S es la suma de todos los naturales comprendidos
 % entre Desde y Hasta, ambos incluidos, avanzando de dos en dos.

 suma_aux(Desde, Hasta, 0) :-
    Desde > Hasta,
    !.

 suma_aux(Desde, Hasta, S) :-
    NDesde is Desde + 2,
    suma_aux(NDesde, Hasta, NS),
    S is NS + Desde.

%% Implementación con recursión final (recursión de cola)

 sumap(N, S) :-
    integer(N),
    N >= 0,
    suma_aux(2, N, 0, S). % el 3er parámetro es el de acumulación
```

```
 % caso base: se devuelve lo acumulado
 suma_aux(Desde, Hasta, Acc, Acc) :-
    Desde > Hasta,
    !.

% caso recursivo: se actualiza el acumulador
 suma_aux(Desde, Hasta, Acc, S) :-
    NDesde is Desde + 2,
    NAcc is Acc + Desde,
    suma_aux(NDesde, Hasta, NAcc, S).
```

4. (2 puntos) Suponga que dispone de los predicados `nota(?Al,?As,?N)`, cierto si `N` es la nota del alumno `Al` en la asignatura `As`, `curso(?As,?N)`, cierto si la asignatura `As` se imparte en el curso `N` ($N \in \{1, \ldots, 4\}$) y `optativas(?L)`, cierto si `L` es una lista con los nombres de las asignaturas optativas. Teniendo en cuenta lo anterior, implemente en Prolog el predicado `max(+Al,?M)`, cierto si `M` es la máxima nota obtenida por `Al` en asignaturas no optativas de los dos últimos cursos (tercero y cuarto). El predicado deberá fallar en caso de que no se disponga de ninguna nota en asignaturas con esas características para el alumno dado. Por ejemplo, suponga disponibles los siguientes datos:

```
nota(a1, as1, 8).          |       curso(as1, 1).
nota(a1, as2, 7).          |       curso(as2, 4).
nota(a1, as3, 10).         |       curso(as3, 3).
nota(a2, as3, 7).          |       optativas([as3]).
```

Con la información anterior, la consulta `?- max(a1,M)` debería dar `M=7` como única respuesta, mientras que `?- max(a2,M)` debería fallar.

Solución:

```
 max(Al, M) :-
       optativas(LO),
   % setof falla si la lista es vacía
       setof(N, As^(nota(Al, As, N), \+ member(As, LO),
               (curso(As,3) ; curso(As,4))), LNotas),
   % setof devuelve la lista ordenada de menor a mayor
       append(_, [M], LNotas).
```

5. (2 puntos) Dados el programa que se incluye en la otra cara de esta hoja y la consulta

```
?- r(X), s(X,Y).
```

dibuje el Árbol de Resolución correspondiente, etiquetando cada arco con el unificador de máxima generalidad utilizado y marcando claramente las posibles ramas podadas, ramas fallo, ramas infinitas o ramas éxito. En caso de haber soluciones, estas solo se valorarán si se acompañan de los cálculos que permiten obtenerlas. Indique además qué respuesta(s) ofrecería Prolog ante la consulta dada, y en qué orden lo haría.

```
p(1).                     |     r(X) :- p(X), \+ q(X).
p(2).                     |     s(X,3) :- X =< 2.
q(2).                     |
```

Solución:

El árbol de Resolución correspondiente (en el examen había que dibujarlo) tiene, recorriéndolo en profundidad, las siguientes ramas: rama fallo, rama éxito con solución `X=1,Y=3`, rama fallo y rama podada por un corte.

Además de dibujar el árbol, de acuerdo con el enunciado, era necesario:

- Acompañar cada una de las soluciones de los cálculos que permiten obtenerlas (aplicando a cada una de las variables de la consulta los umg's de la rama, en orden, empezando por la raíz).
- Indicar qué respuestas ofrece Prolog ante la consulta dada y en qué orden las da: dado que Prolog construye el árbol en profundidad por la izquierda, ignorando tanto ramas fallo como ramas podadas, la única respuesta sería `X=1,Y=3` y a continuación `false` (indicando que no hay más soluciones).

Universidad Rey Juan Carlos

Prueba de Programación Lógica M-A

20-11-2020

Apellidos: ______________________ **Nombre:** ______________

Grado: GII / GII-ONLINE / GII+GADE / GII+GIC / GII+GIS / GII+MAT

- La duración de esta prueba es de 1 hora y 15 minutos.
- No se permite el uso de ningún tipo de material ni dispositivo electrónico.

Además de los predicados aritméticos, el corte y la negación, se podrán utilizar los siguientes: `length(?L,?N)` (cierto si `N` es la longitud de la lista `L`), `append(?L1,?L2,?L)` (cierto si `L` es la concatenación de las listas `L1` y `L2`), `reverse(?L1,?L2)` (cierto si `L2` es la inversa de la lista `L1`), `member(?E,?L)` (cierto si `E` pertenece a `L`), `is_list(+L)` (cierto si `L` es una lista), `sumlist(+L,?N)` (cierto si `L` es una lista de números cuya suma es `N`), `map(+Obj,+L)` (cierto si `Obj` se ejecuta con éxito sobre todos los elementos de la lista `L`), `map(+Obj,?L1,?L2)` (cierto si `Obj` se ejecuta con éxito sobre todas las parejas de elementos de las listas `L1` y `L2` situados en la misma posición), `include/exclude(+Obj,+L1,?L2)` (cierto si `L2` incluye/excluye los elementos de la lista `L1` sobre los que se ejecuta con éxito `Obj`), `pliegai(+Obj,+L,+VI,-VF)`, cierto si `VF` es el resultado de plegar mediante `Obj` la lista `L` desde la izquierda partiendo de `VI`), `number/integer/var/nonvar(+N)` (cierto si `N` es un número/entero/variable/novariable), `call(+Obj,+E1,..)` (cierto si `Obj`, con parámetros extra `E1,..`, se ejecuta con éxito) y los predicados de recolección `bagof/setof/findall(?T, +Obj, ?L)`.

1. (1.5 puntos) Sabiendo que dispone de los predicados `gusta(X,Y)`, cierto si a `X` le gusta el lenguaje `Y` y `declarativo(X)`, cierto si el lenguaje `X` es declarativo, proponga una implementación para el predicado `ex(+N,?L)`, cierto si `N` es un natural positivo y `L` es la lista de personas a las que les gustan al menos `N` lenguajes declarativos distintos. El predicado debe fallar si no hubiese ninguna persona con las características pedidas.

Solución:

```
ex(N, L) :-
 integer(N),
 N > 0,
 setof(P,
       LL^Tam^(setof(Leng, (gusta(P, Leng), declarativo(Leng)), LL),
              length(LL, Tam),
              Tam >= N),
       L).
```

2. Considere el predicado `ex(+L, +E, ?NL)`, cierto si `L` es una lista con al menos una ocurrencia del elemento `E` y `NL` es la lista conteniendo exclusivamente los elementos de `L` situados a la izquierda de la primera ocurrencia de `E` que resultan ser números y cuyos cuadrados no pertenecen a `L`. El predicado no debe producir ningún error en tiempo de ejecución y debe fallar si `L` no cumple las condiciones dadas. Por ejemplo, las consultas `?- ex(a,a,X)`, `?- ex([],a,X)` y `?- ex([1,a],*,X)` deben devolver `false`, y `?- ex([a,2,2,*,4,b], *, X)` y `?- ex([A,3,a,2,3,4,5,*,25,7,*],*,X)` deben devolver, respectivamente, `X=[]` y `X=[3,3,4]` como única solución.

   (a) (2 puntos) Proponga una implementación **no recursiva** para el predicado `ex`, basada en el uso de algunos de los predicados mencionados al comienzo del enunciado.

   Solución:

```
examen(L, E, NL) :-
    append(Pref, [E|_], L), % contiene E
    !,  % solo interesa la primera ocurrencia de E
    findall(X, (member(X,Pref), number(X),
                X2 is X*X, \+ member(X2, L)), NL).
```

   (b) (2 puntos) Proponga una implementación **recursiva** para el predicado `ex`, en la que no podrá usar ni predicados de recolección ni predicados de aplicación (maplist) o filtrado (include/exclude).

   Solución:

```
ex(L, E, NL) :-
    aux(L, L, E, NL).
aux([E|R], L, E, []) :-
    !.
aux([C|R], L, E, [C|NR]) :-
    number(C),
    C2 is C*C,
    \+ member(C2, L),
    !,
    aux(R,L,E,NR).
aux([_|R], L, E, NR) :-
    aux(R,L,E,NR).
```

3. Considere el siguiente código:

```
mist(L, E, N) :-
```

```
    mist(L, E, 0, N).
mist([], _, N, N).
mist([E|R], E, A, N) :-
    !,
    NA is A + 1,
    mist(R, E, NA, N).
mist([_|_], _, A, A).
```

(a) (1 punto) Describa en lenguaje natural (con sus propias palabras) el cometido del predicado `mist/3`: "`mist(L,E,N)` es cierto si ..."

> **Solución:**
> `mist(L,E,N)` es cierto si N es el tamaño del prefijo más largo de la lista L conteniendo solo elementos unificables con E.

(b) (1 punto) ¿Presenta el código anterior recursión de cola (recursión final)? Justifique su respuesta y proponga una implementación alternativa, con o sin recursión de cola dependiendo de su respuesta a la pregunta anterior.

> **Solución:**
> El código anterior incorpora un parámetro de acumulación para conseguir recursión de cola (en la única cláusula recursiva, la llamada recursiva se realiza en último lugar). Una implementación alternativa sin parámetro de acumulación, que deja de tener recursión de cola, es la siguiente:

```
mist([], _, 0).
mist([E|R], E, N) :-
    !,
    mist(R, E, NR),
    N is NR + 1.
mist([_|_], _, 0).
```

4. (2.5 puntos) Dados el programa que se incluye en la otra cara de esta hoja y la consulta

```
?- r(U), \+ p(U), r(V).
```

dibuje el Árbol de Resolución correspondiente, etiquetando cada arco con el unificador de máxima generalidad utilizado y marcando claramente las posibles ramas podadas, ramas fallo, ramas infinitas o ramas éxito. En caso de haber soluciones, estas solo se valorarán si se acompañan de los cálculos que permiten obtenerlas. Indique además qué respuesta(s) ofrecería Prolog ante la consulta dada, y en qué orden lo haría.

```
p(f(X)) :- p(X).             |          r(f(a)).
p(f(a)).                     |          r(a) :- !.
                             |          r(b).
```

Solución:

El árbol de Resolución correspondiente (en el examen había que dibujarlo) tiene, recorriéndolo en profundidad, las siguientes ramas: dos ramas fallo, una rama podada por un corte, una rama fallo, una rama éxito con solución `U=a, V=f(a)`, otra rama éxito con solución `U=a, V=a` y por último dos ramas podadas por cortes.

Además de dibujar el árbol, de acuerdo con el enunciado, era necesario:

- Acompañar cada una de las soluciones de los cálculos que permiten obtenerlas (aplicando a cada una de las variables de la consulta los umg's de la rama, en orden, empezando por la raíz).
- Indicar qué respuestas ofrece Prolog ante la consulta dada y en qué orden las da: dado que Prolog construye el árbol en profundidad por la izquierda, ignorando tanto ramas fallo como ramas podadas, la primera respuesta sería `U=a, V=f(a)`, la segunda respuesta sería `U=a, V=a` y acabaría con un `false` (indicando que no hay más soluciones).

Prueba de Programación Lógica V-A

18-11-2020

Apellidos: ____________________ **Nombre:** ____________

Grado: GII / GII+GADE / GII+GCRIM

- La duración de esta prueba es de 1 hora y 15 minutos.
- No se permite el uso de ningún tipo de material ni dispositivo electrónico.

Además de los predicados aritméticos, el corte y la negación, se podrán utilizar los siguientes: `length(?L,?N)` (cierto si `N` es la longitud de la lista `L`), `append(?L1,?L2,?L)` (cierto si `L` es la concatenación de las listas `L1` y `L2`), `reverse(?L1,?L2)` (cierto si `L2` es la inversa de la lista `L1`), `member(?E,?L)` (cierto si `E` pertenece a `L`), `is_list(+L)` (cierto si `L` es una lista), `sumlist(+L,?N)` (cierto si `L` es una lista de números cuya suma es `N`), `map(+Obj,+L)` (cierto si `Obj` se ejecuta con éxito sobre todos los elementos de la lista `L`), `map(+Obj,?L1,?L2)` (cierto si `Obj` se ejecuta con éxito sobre todas las parejas de elementos de las listas `L1` y `L2` situados en la misma posición), `include/exclude(+Obj,+L1,?L2)` (cierto si `L2` incluye/excluye los elementos de la lista `L1` sobre los que se ejecuta con éxito `Obj`), `pliegai(+Obj,+L,+VI,-VF)`, cierto si `VF` es el resultado de plegar mediante `Obj` la lista `L` desde la izquierda partiendo de `VI`), `number/integer/var/nonvar(+N)` (cierto si `N` es un número/entero/variable/novariable), `call(+Obj,+E1,..)` (cierto si `Obj`, con parámetros extra `E1,..`, se ejecuta con éxito) y los predicados de recolección `bagof/setof/findall(?T, +Obj, ?L)`.

1. (1.5 puntos) **Razone** qué se obtendría (un error, `true`, `false`, computación infinita, respuesta(s) que sería(n) ...) al ejecutar en Prolog la siguiente consulta, solicitando todas las posibles soluciones:

```
?- findall(NY, (append(T,Y,[1,2,a,4]),
                length(Y,LY),
                LY >= 2, include(integer,Y,NY)), R),
   map(sumlist, R, NR),
   member(A, NR), !.
```

Solución:

```
R = [[1, 2, 4], [2, 4], [4]],
NR = [7, 6, 4],
A = 7.
```

2. (2 puntos) Considere el predicado `ex(+L, +N, ?NL)`, cierto si `L` es una lista compuesta exclusivamente por términos no variables, `N` es un natural positivo y `NL` es

la lista resultante tras duplicar en `L` el elemento situado en la posición `N`, entendiendo que la cabeza de la lista está situada en la posición 1. El predicado no debe producir ningún error en tiempo de ejecución y debe fallar si la operación no se puede ejecutar porque la posición `N` es mayor que la longitud de `L`. Por ejemplo, las consultas `?- ex(a,1,X)`, `?- ex([1],-1,X)`, `?- ex([],1,X)`, `?- ex([2],3,X)`, `?- ex([A],1,X)` y `?- ex([1,2],a,X)` darían `false`, `?- ex([1,2],1,[1,1,2])` devolvería `true` y `?- ex([1,a,5,3],3,X)` daría `X=[1,a,5,5,3]` como única solución. Proponga una implementación **no recursiva** para el predicado `ex`, basada en el uso de algunos de los predicados mencionados al comienzo del enunciado.

Solución:

```
ex(L, N, NL) :-
    maplist(nonvar, L),
    integer(N),
    N >= 1,
    append(Pref, [C|R], L),
    N1 is N-1,
    length(Pref, N1),
    append(Pref, [C,C|R], NL).
```

3. (2 puntos) Considere el predicado `ex(+L, ?NL)`, cierto si `L` es una lista y `NL` es la lista obtenida de `L` tras intercambiar el último número (empezando por la izquierda) que aparece en `L` con el elemento situado justo a continuación suyo. Si `L` no contuviese ningún número o su último número no tuviese siguiente, `NL` sería igual a `L`. Por ejemplo, la consulta `?- ex(a,X)` daría `false` y `?- ex([a,b],X)`, `?- ex([1,a,4],X)`, `?- ex([1,3,a],X)` y `?- ex([a,1,3,c,a],X)` darían, respectivamente, `X=[a,b]`, `X=[1,a,4]`, `X=[1,a,3]` y `ex([a,1,c,3,a],X)` como única solución. Proponga una implementación **recursiva** para el predicado `ex`.

Solución:

```
ex([], []).
ex([C1,C2|R], [C2,C1|R]) :-
    number(C1),
    findall(X, (member(X,[C2|R]), number(X)), []),
    !.
ex([C|R], [C|NR]) :-
    ex(R, NR).
```

4. Considere el siguiente código:

```
mist([], 0).
mist([C|R], N) :-
    number(C),
    !,
    mist(R, N).
mist([_|R], N):-
    mist(R, NR),
    N is NR+1.
```

(a) (0.5 puntos) Describa en lenguaje natural (con sus propias palabras) el cometido del predicado anterior: “`mist(L,N)` es cierto si ...”

Solución:
`mist(L,N)` es cierto si N es el número de elementos de la lista L que no son números.

(b) (1.5 puntos) ¿Presenta el código anterior recursión de cola (recursión final)? Justifique su respuesta y proponga una implementación alternativa, con o sin recursión de cola dependiendo de su respuesta a la pregunta anterior.

Solución:
La implementación propuesta no tiene recursión de cola puesto que en la tercera cláusula se realiza una suma *después* de la llamada recursiva. Una implementación alternativa, incorporando un parámetro de acumulación para conseguir recursión de cola, podría ser la siguiente:

```
mist_c(L, N) :-
    mist_c(L, 0, N).

mist_c([], Ac, Ac).
mist_c([C|R], Ac, N) :-
    number(C),
    !,
    mist_c(R, Ac, N).
mist_c([_|R], Ac, N) :-
    NAc is Ac + 1,
    mist_c(R, NAc, N).
```

5. (2.5 puntos) Dados el programa que se incluye en la otra cara de esta hoja y la consulta

```
?- - q(X,Y), \+ m(X).
```

dibuje el Árbol de Resolución correspondiente, etiquetando cada arco con el unificador de máxima generalidad utilizado y marcando claramente las posibles ramas podadas, ramas fallo, ramas infinitas o ramas éxito. En caso de haber soluciones, estas solo se valorarán si se acompañan de los cálculos que permiten obtenerlas. Indique además qué respuesta(s) ofrecería Prolog ante la consulta dada, y en qué orden lo haría.

```
q(X,Y) :- m(X), h(X,Y).        |        h(b,c).
q(X,Y) :- h(Y,X).              |        m(b).
q(a,b) :- !.                   |
q(a,c).                        |
```

Solución:

Universidad Rey Juan Carlos

Prueba de Programación Lógica M-A

21-10-2019

Apellidos: ______________________ **Nombre:** ______________

Grado: GII / GII-ONLINE / GII+GADE / GII+GIC / GII+GIS / GII+MAT

- La duración de esta prueba es de 1 hora y 30 minutos.
- No se permite el uso de ningún tipo de material ni dispositivo electrónico.

Además de los predicados aritméticos, el corte y la negación, se podrán utilizar los siguientes: `length(?L,?N)` (cierto si `N` es la longitud de la lista `L`), `append(?L1,?L2,?L)` (cierto si `L` es la concatenación de las listas `L1` y `L2`), `member(?E,?L)` (cierto si `E` pertenece a `L`), `sumlist(+L,?N)` (cierto si `L` es una lista de números cuya suma es `N`), `map(+Obj,+L)` (cierto si `Obj` se ejecuta con éxito sobre todos los elementos de la lista `L`), `map(+Obj,?L1,?L2)` (cierto si `Obj` se ejecuta con éxito sobre todas las parejas de elementos de las listas `L1` y `L2` situados en la misma posición), `include/exclude(+Obj,+L1,?L2)` (cierto si `L2` incluye/excluye los elementos de la lista `L1` sobre los que se ejecuta con éxito `Obj`), `pliegai(+Obj,+L,+VI,-VF)`, cierto si `VF` es el resultado de plegar mediante `Obj` la lista `L` desde la izquierda partiendo de `VI`), `number/integer(+N)` (cierto si `N` es un número/entero), `call(+Obj,+E1,..)` (cierto si `Obj`, con parámetros extra `E1,..`, se ejecuta con éxito) y los predicados de recolección `bagof/setof/findall(?T, +Obj, ?L)`.

1. (1 punto) Sabiendo que dispone de los predicados `gusta(X,Y)`, cierto si a `X` le gusta el lenguaje `Y`, `declarativo(X)`, cierto si el lenguaje `X` es declarativo, y `alérgico(X,Y)`, cierto si `X` es alérgico a `Y`, formalice en Lógica de Primer Orden (LPO) y en Prolog las siguientes afirmaciones y preguntas (si considera que alguna no puede escribirse en LPO o en Prolog, razone por qué): *(i) A todo el mundo le gustan los lenguajes declarativos, salvo aquellos que le dan alergia. (ii) ¿Es cierto que no todos los lenguajes que dan alergia a Pepita son declarativos?*

Solución:

(i) LPO: $\forall X \forall Y[(declarativo(Y) \land \neg alérgico(X,Y)) \rightarrow gusta(X,Y)]$
Prolog: `gusta(X,Y) :- declarativo(Y), \+ alérgico(X,Y).`

(ii) LPO: $\exists Y[alérgico(pepita,Y) \land \neg declarativo(Y)]$
Prolog: `?- alergico(pepita,Y), \+ declarativo(Y).`

2. (0,5 puntos) Indique qué ocurriría al ejecutar la siguiente consulta (un error, respuesta false, respuesta true, computación infinita, la o las respuestas serían ....).

```
?- append(P,[X|_],[1,2,3,4,5]),length(P,LP),LP mod 2 =:= 0,member(Y,P),!.
```

Solución:

Prolog devolvería como única respuesta

```
LP = 2,
P = [1, 2],
X = 3,
Y = 1
```

3. (3,5 puntos) Suponga que necesita obtener, por un lado, la lista conteniendo todos los números naturales múltiplos de 3 comprendidos entre 0 y 50 y, por otro, la lista conteniendo todos los años bisiestos entre 2050 y 2100 (recuerde que un año bisiesto es aquel que es múltiplo de 4 y además, o bien no es múltiplo de 100 o bien es múltiplo de 400). Proponga una implementación para los predicados `exA1(?L1)` y `exA2(?L2)`, ciertos si `L1` es la primera de estas listas y `L2` es la segunda, ambas ordenadas de menor a mayor, de forma que las respuestas a las consultas `exA1(L1)` y `exA2(L2)` serían `L1 = [0,3,..,48], L2 = [2052,2056,..,2096]`. Se valorará muy especialmente la *concisión y la facilidad para la reutilización* del código propuesto. Recuerde que el parámetro del predicado de clasificación `integer(+X)` es de entrada, por lo que este predicado *no sirve para generar valores*.

Solución:

```
exA1(L) :-
        listaNumObj(0, 50, multiplo(3), L).
exA2(L) :-
        listaNumObj(2050, 2100, bisiesto, L).

%multiplo(+X,+Y), cierto si Y es múltiplo de X
multiplo(X,Y) :-
        Y mod X =:= 0.

%bisiesto(+X), cierto si X es un año bisiesto
bisiesto(X) :-
    multiplo(4,X), (\+ multiplo(100,X) ; multiplo(400,X)).


%listaNumObj(+Inf, +Sup, +Obj, ?L)
%cierto si Inf y Sup son números enteros tq Inf =< Sup
%y L es la lista, ordenada de menor a mayor, de los enteros
%comprendidos entre Inf y Sup sobre los que se puede aplicar
%Obj con éxito.
```

```
listaNumObj(Inf, Sup, Obj, L) :-
        listaNum(Inf, Sup, LNum),
        include(Obj, LNum, L).

%listaNum(+Inf, +Sup, ?L)
%cierto si Inf y Sup son números enteros tq Inf =< Sup
%y L = [Inf, Inf+1, ..., Sup]

 listaNum(Inf, Sup, L) :-
    integer(Inf),
    integer(Sup),
    Inf =< Sup,
    entre(Inf, Sup, L).

 entre(Inf, Inf, L) :-
        !,
        L = [Inf].

 entre(Inf, Sup, [Inf|R]) :-
    NInf is Inf+1,
    entre(NInf, Sup, R).
```

4. (3 puntos) Suponga disponibles los predicados `ingresos(?X,?S,?I)`, cierto si `I` son los ingresos (en miles de euros/mes) de `X` por su trabajo en el sector `S`, `baja(?X)`, cierto si `X` está de baja y `s_min(?C)`, cierto si `C` es el salario mínimo vigente (en miles de euros/mes). Proponga una implementación para el predicado `porcentaje(+S,?P)`, cierto si `P` es el porcentaje, sobre el total de trabajadores del sector `S`, de aquellos que no están de baja y cuyos ingresos son superiores a tres veces el salario mínimo. Se deberá tener en cuenta que una misma persona puede tener varios trabajos (incluso en el mismo sector y con el mismo salario) y que puede no haber trabajadores en el sector pedido, en cuyo caso se devolverá `false`. Por ejemplo, con los datos:

```
ingresos(pepa,'educación',1.5).   |    ingresos(pepa,'educación',1.5).
ingresos(pepe,'educación',3).     |    ingresos(pepito,'servicios',2).
s_min(0.9).                       |    baja(pepe).
```

`porcentaje('educación',P)` y `porcentaje('servicios',P)` darían `P=50` (1 de 2) y `P=0` (0 de 1), mientras que `porcentaje('informática',P)` daría `false`.

Solución:

```
porcentaje(S, P) :-
%trabajadores totales del sector (setof para no repetir
```

```
% aquellos con varios contratos y fallar si no hay)
    setof(X, I^(ingresos(X, S, I)), LTs),
    s_min(M),
% trabajadores del sector con las condiciones pedidas
% (findall para que no falle si no hay)
    findall(X, (member(X,LTs),
                \+ baja(X),
                bagof(Ing, ingresos(X,S,Ing), LI),
                % (o findall, puede haber salarios iguales)
                sumlist(LI, I),
                I > 3*M),
            LTPs),
    length(LTs, CuantosTs),
    length(LTPs, CuantosTPs),
    P is 100*CuantosTPs/CuantosTs.
```

5. (2 puntos) Dados el programa que se incluye en la otra cara de esta hoja y la consulta

   `?- s(B,A), \+ q(A,B).`

   dibuje el Árbol de Resolución correspondiente, marcando la(s) posible(s) rama(s) podada(s), e indique qué respuesta(s) ofrece Prolog y en qué orden lo hace.

```
q(b,a).                     |       s(b,c).
q(X,Y) :- r(X).             |       s(X, Y) :- s(Y, X), !.
r(b).                       |       s(X, Y) :- q(X, X).
```

Solución:

El árbol de Resolución correspondiente (en el examen había que dibujarlo) tiene, recorriéndolo en profundidad por la izquierda, las siguientes ramas: una rama fallo, una rama éxito con solución asociada `B=b, A=c`, una rama fallo y cuatro ramas podadas por cortes.

Por lo tanto, Prolog devuelve una única respuesta, `B=b, A=c`, y a continuación `false`, indicando que no hay más respuestas.

Universidad Rey Juan Carlos

Prueba de Programación Lógica V-A

28-10-2019

Apellidos: ______________________ **Nombre:** ______________

Grado: GII / GII-ONLINE / GII+GADE / GII+GIC / GII+GIS / GII+MAT

- La duración de esta prueba es de 1 hora y 30 minutos.
- No se permite el uso de ningún tipo de material ni dispositivo electrónico.

Además de los predicados aritméticos, el corte y la negación, se podrán utilizar los siguientes: `length(?L,?N)` (cierto si `N` es la longitud de la lista `L`), `append(?L1,?L2,?L)` (cierto si `L` es la concatenación de las listas `L1` y `L2`), `member(?E,?L)` (cierto si `E` pertenece a `L`), `sumlist(+L,?N)` (cierto si `L` es una lista de números cuya suma es `N`), `map(+Obj,+L)` (cierto si `Obj` se ejecuta con éxito sobre todos los elementos de la lista `L`), `map(+Obj,?L1,?L2)` (cierto si `Obj` se ejecuta con éxito sobre todas las parejas de elementos de las listas `L1` y `L2` situados en la misma posición), `include/exclude(+Obj,+L1,?L2)` (cierto si `L2` incluye/excluye los elementos de la lista `L1` sobre los que se ejecuta con éxito `Obj`), `pliegai(+Obj,+L,+VI,-VF)`, cierto si `VF` es el resultado de plegar mediante `Obj` la lista `L` desde la izquierda partiendo de `VI`), `number/integer(+N)` (cierto si `N` es un número/entero), `call(+Obj,+E1,..)` (cierto si `Obj`, con parámetros extra `E1,..`, se ejecuta con éxito) y los predicados de recolección `bagof/setof/findall(?T, +Obj, ?L)`.

1. (0,5 puntos) Recuerde que el predicado `suma(?X,?Y,?Z)` estudiado en clase es un predicado lógico puro, implementado sin usar la aritmética de Prolog, cierto si `Z` es la suma de `X` e `Y`. El predicado trabaja con números naturales representados mediante `0, s(0), s(s(0)),..`, aunque en este ejercicio, por comodidad, se usará la notación `s^i(0)` para denotar al número $i \in \{2, 3, \ldots\}$. Indique qué se obtendría (un error, respuesta false, respuesta true, una computación infinita, respuesta(s) que sería(n) ...) al ejecutar en Prolog la siguiente consulta.

```
?- setof(X, U^V^(suma(X,U,s^4(0)), suma(U,V,X), X\=U), LX),
   append(P, [_], [0|LX]), member(A,P), !.
```

Solución:

`LX = [s^3(0), s^4(0)],P = [0,s^3(0)], A=0`
y false indicando que no hay más soluciones.

2. Considere el predicado `el(+L, +Pos, +N, ?NL)`, cierto si `NL` es la lista resultante tras eliminar de `L` tantos elementos consecutivos como indique `N` empezando en la posición `Pos`, entendiendo que la cabeza de la lista está situada en la posición 1. Tanto `Pos` como `N` deben ser naturales positivos. El predicado no debe producir ningún error en tiempo de ejecución y debe fallar si la operación no se puede

ejecutar porque la posición `Pos` es mayor que la longitud de `L` o porque `N` es mayor que el número de elementos que hay disponibles en `L` contando desde `Pos`. Por ejemplo, las consultas `?- el([],1,1,X)`, `?- el([1],0,1,X)`, `?- el([1],3,1,X)` y `?- el([1,2],2,2,X)` deben devolver `false`, `?- el([1,2],1,1,[2])` devolvería `true` y las consultas `?- el([1,2,3,4,5],1,3,X)`, `?- el([4,5,1],3,1,X)` y `?- el([4,1,2,3,5],2,3,X)` deben devolver `X=[4,5]` como única solución.

(a) (1,5 puntos) Proponga una implementación *recursiva* para el predicado `el`.

Solución:

```
el(L, Pos, N, NL) :-
    maplist(positivo, [Pos,N]),
    el1(L, Pos, N, NL).

el1([_|R], 1, 1, R) :- !.

el1([_|R], 1, N, NR) :-
    !,
    N1 is N - 1,
    el1(R, 1, N1, NR).

el1([C|R], Pos, N, [C|NR]) :-
    Pos1 is Pos - 1,
    el1(R, Pos1, N, NR).

positivo(X) :-
        integer(X),
        X >= 1.
```

(b) (1,5 puntos) Proponga una implementación *no recursiva* para el predicado `el`, basada en el uso de algunos de los predicados mencionados al comienzo del enunciado.

Solución:

```
el_bis(L, Pos, N, NL) :-
    maplist(positivo, [Pos,N]),
    Pos1 is Pos - 1,
    append(Pre, Suf, L),
    length(Pre, Pos1),
    append(Suf1, Suf2, Suf),
    length(Suf1, N),
    append(Pre, Suf2, NL).
```

☐

3. (3 puntos) Suponga disponibles los predicados `cp(?CP,?P,?H)`, cierto si `CP` es una capital de provincia del país `P` con `H` millones de habitantes y `caps(?LC)`, cierto si `LC` es una lista con las capitales de provincia que son, además, capitales de sus países. Proponga una implementación para el predicado `mega_caps(-LP,-Por)`, cierto si `LP` es una lista conteniendo aquellos países cuyas capitales albergan más población que el resto de capitales de provincia del país juntas, y `Por` es el porcentaje de países que cumplen lo anterior. Se supondrá que hay al menos un país y que todos los países tienen al menos dos capitales de provincia, siendo una de ellas la capital del país. Por ejemplo, con los datos:

```
cp(c11,p1,6).  |  cp(c12,p1,2).  |   cp(c13,p1,2).  |  caps([c11,c21]).
cp(c21,p2,4).  |  cp(c22,p2,3).  |   cp(c23,p2,2).
```

la respuesta a la consulta `mega_caps(LP,Por)` sería `LP=[p1], Por=50`. Si en los datos anteriores se cambiase el 6 por un 3, la respuesta sería `LP=[], Por=0`.

Solución:

```
mega_caps(PaisesConMegaCap, Por) :-
    caps(Capitales),
    length(Capitales, NumPaises),
    findall(Pais, (cp(Cap, Pais, HabsCap), % (lista vacía si no hay)
                   member(Cap, Capitales),
                   findall(HCP, (cp(CP, Pais, HCP), CP \= Cap),
                           HabsCPs),
                   sumlist(HabsCPs, HabsCP),
                   HabsCap > HabsCP),
            PaisesConMegaCap),
    length(PaisesConMegaCap, NumPaisesConMegaCap),
    Por is 100*NumPaisesConMegaCap/NumPaises.
```

4. Considere el predicado `a` descrito en la otra cara de esta hoja.
   (a) (2 puntos) Dados `q(X) :- \+ X mod 3 =:= 0.` y la consulta `?- a(q,0,2,V)`, dibuje el Árbol de Resolución correspondiente, marcando la(s) posible(s) rama(s) podada(s), e indique qué respuesta(s) ofrece Prolog y en qué orden lo hace.
   (b) (1,5 puntos) ¿Podría `a` producir algún error? En caso afirmativo, razone cúal(es) y por qué. Indique también qué ocurriría (error, true, false, la(s) respuesta(s) sería(n), ...) si se usase `a` con todos los argumentos de entrada salvo el último.

Solución:

- Los parámetros I,S son utilizados por predicados aritméticos, por lo que si no son expresiones aritméticas evaluables producirán errores de aritmética o de instanciación.
- El parámetro P se utiliza como primer argumento de un call, por lo que se produciría un error si, trás añadirle como argumento el segundo parámetro del call, no fuese un predicado ejecutable.
- Si se usa `a/4` con todos los argumentos de entrada salvo el último, al que se le pasa una variable, o bien se produce alguno de los errores mencionados más arriba, o bien las respuestas serían los números comprendidos entre I y S, empezando en I y separados entre ellos por una unidad, sobre los que se aplica con éxito el objetivo P.

```
a(P,I,S,I) :- I =< S, call(P, I).
a(P,I,S,V) :- I < S, NI is I+1, a(P,NI,S,V).
```

Solución:
El árbol de Resolución correspondiente (en el examen había que dibujarlo) tiene, recorriéndolo en profundidad por la izquierda, las siguientes ramas: una rama fallo, una rama podada, una rama fallo, una rama éxito con solución asociada `V=1`, una rama fallo, una rama éxito con solución asociada `V=2` y una rama fallo.

Por lo tanto, Prolog devuelve como primera solución `V=1`, como segunda solución `V=2` y a continuación `false`, indicando que no hay más respuestas.

Universidad Rey Juan Carlos **Prueba de Programación Lógica M-A** 13-6-2019

Apellidos: ______________________ **Nombre:** ______________

Grado: GII / GII-ONLINE / GII+GADE / GII+GIC / GII+GIS / GII+MAT

- La duración de esta prueba es de 1 hora y 20 minutos.
- No se permite el uso de ningún tipo de material ni dispositivo electrónico.

Se supondrán disponibles los siguientes predicados: `length(?L,?N)` (cierto si N es la longitud de la lista L), `append(?L1,?L2,?L)` (cierto si L es la concatenación de las listas L1 y L2), `member(?E,?L)` (cierto si E pertenece a L), `sumlist(+L,?N)` (cierto si `L` es una lista de números cuya suma es N), `map(+P,+L)` (cierto si `L` es una lista tal que todos sus elementos cumplen la propiedad `P/1`), `map(+P,?L1,?L2)` (cierto si `P(x_i,y_i)` se cumple para todos los pares `(x_i,y_i)` donde `x_i` e `y_i` son los iésimos elementos de L1 y L2), `include/exclude(+P,+L1,?L2)` (cierto si `L2` está formada por los elementos de la lista L1 que cumplen/no cumplen la propiedad `P/1`), `number/integer/natural(+N)` (cierto si `N` es un número/entero/natural), `?T =.. ?L` (cierto si *T = fun(a1,..,an)* y *L=[fun,a1,..,an]*), `call(+Obj)` (cierto si el objetivo Obj se ejecuta con éxito) y los predicados de recolección de soluciones `bagof(?T, +Obj, ?L)`, `setof(?T, +Obj, ?L)` y `findall(?T, +Obj, ?L)`.

1. (0.5 puntos) Indique qué respuesta(s) ofrecería Prolog ante la siguiente consulta:

```
?- a(U,4) = a(V,V), append(L, _, [1,2|[3,U]]),
   length(L,LL), 0 =:= LL mod 2.
```

> **Solución:**
> Prolog devolvería tres respuestas:
>
> ```
> U=V=4, L = [], LL = 0
> U=V=4, L = [1, 2], LL = 2
> U=V=4, L = [1, 2, 3, 4], LL = 4
> ```
>
> y acabaría con un `false` para indicar que no hay más soluciones.

2. Considere el ejemplo estudiado en clase relativo al *mundo de los bloques*, y suponga disponibles los predicados `encima(X,Y)`, cierto si el bloque `X` está justo encima del bloque `Y`, `apilado(X,Y)`, cierto si `X` está apilado sobre `Y` y `sobre_mesa(X)`, cierto si `X` está colocado sobre la mesa (no tiene ningún bloque debajo).

(a) (1 punto) Implemente el predicado `altura(?B,?A)`, cierto si `A` es la altura del bloque `B` en su pila, entendiendo que el bloque colocado sobre la mesa tiene altura 1, el bloque encima de este último tiene altura 2, etc.

Solución:

```
altura(B, 1) :-
    sobre_mesa(B).
altura(B, A) :-
    encima(B,C),
    altura(C,AC),
    A is AC + 1.
```

(b) (2 puntos) Implemente el predicado `ap(+B,+N,?L)`, cierto si `L` es una lista conteniendo todas las tuplas `(X,AX)` donde `X` es un bloque que está apilado sobre el bloque `B`, no está justo encima suyo y cuya altura `AX` no es inferior a `N`. `L` deberá ser la lista vacía si no hubiese ningún bloque con las características requeridas. Por ejemplo, suponiendo que el bloque `a` está sobre la mesa y que se tiene la pila `abcd`, `ap(a,1,L)` y `ap(a,2,L)` darían ambas `L=[(c,3),(d,4)]`, mientras que `ap(a,4,L)` y `ap(c,2,L)` darían `L=[(d,4)]` y `L=[]`, respectivamente.

Solución:

```
ap(B, N, L) :-
    findall((X,AX),
        (apilado(X,B), \+ encima(X,B),
         altura(X,AX), AX >= N), L).
```

3. (3 puntos) Proponga una implementación para el predicado `dobla(+L, +N, ?NL)`, cierto si `L` es una lista de números, `N` es un número natural y `NL` es la lista conteniendo los mismos elementos que `L` y en el mismo orden pero tal que sus primeros `N` elementos se reemplazan por sus cuadrados. Si `N` fuese cero la lista no se modificaría, y si `L` tuviese menos de `N` elementos se reemplazarían todos. El predicado no debe producir ningún error en tiempo de ejecución. Por ejemplo, las consultas `dobla([2,3],2,NL)` y `dobla([2,3],4,NL)` darían ambas `NL=[4,9]`, mientras que `dobla([2,3,4],1,NL)`, `dobla([],3,NL)` y `dobla([2,3],0,NL)` darían, respectivamente, `NL=[4,3,4]`, `NL=[]` y `NL=[2,3]`.

Solución:

```
dobla(L, N, NL) :-
    maplist(number,L),
```

```
        natural(N),
        dobla1(L,N,NL).

    dobla1([], _, []).
    dobla1(L, 0, L) :- !.
    dobla1([C|R], N, [CC|NR]) :-
       N1 is N - 1,
       CC is C*C,
       dobla1(R, N1, NR).
```

4. Considere el programa Prolog descrito en la otra cara de esta hoja.

   (a) (1.5 puntos) Describa en lenguaje natural (con sus propias palabras) el *resultado* del predicado `q` (`q(X,P,C)` es cierto si ...) y razone si podría producir algún error en tiempo de ejecución y si el corte de la primera cláusula tiene alguna utilidad (¿tendría algún efecto su eliminación?).

   Solución:
   `q(X,P,C)` devuelve cierto si C es el primer elemento de la lista X situado en una posición impar (suponiendo que se empieza a contar por 1 desde la izquierda) que cumple la propiedad P/1. Se podría producir un error de ejecución si se pasase como parámetro `P` algún predicado no definido, con aridad distinta de uno o cuya ejecución produzca algún error. Si se suprimiese el corte las dos cláusulas dejarían de ser incompatibles por lo que el predicado produciería varias soluciones, asociando a C *todos* los elementos situados en posiciones impares de la lista X que cumplen P.

   (b) (2 puntos) Dada la consulta

   ```
   ?- q([a,2,4,3], integer, E), \+ i(E).
   ```

   dibuje el Árbol de Resolución correspondiente, marcando la(s) posible(s) rama(s) podada(s), e indique qué respuesta(s) ofrece Prolog y en qué orden lo hace.

   ```
   q([C|R], P, C) :- O =.. [P, C], call(O), !.
   q([_C1,_C2|R], P, C) :- q(R, P, C).
   i(X) :- X mod 2 =:= 1.
   ```

   Solución:
   El árbol de Resolución correspondiente (en el examen había que dibujarlo) tiene, recorriéndolo en profundidad por la izquierda, las siguientes ramas: dos ramas fallo, una rama éxito con solución asociada `E=4` y una rama podada por un corte.

   Por lo tanto, Prolog devuelve una única respuesta, `E=4`, y a continuación `false`, indicando que no hay más respuestas.

Prueba de Programación Lógica V-A 13-6-2019

Apellidos: ______________________ **Nombre:** ______________

Grado: GII / GII-ONLINE / GII+GADE / GII+GIC / GII+GIS / GII+MAT

- La duración de esta prueba es de 1 hora y 20 minutos.
- No se permite el uso de ningún tipo de material ni dispositivo electrónico.

Se supondrán disponibles los siguientes predicados: `length(?L,?N)` (cierto si N es la longitud de la lista L), `append(?L1,?L2,?L)` (cierto si L es la concatenación de las listas L1 y L2), `member(?E,?L)` (cierto si E pertenece a L), `sumlist(+L,?N)` (cierto si `L` es una lista de números cuya suma es N), `map(+P,+L)` (cierto si `L` es una lista tal que todos sus elementos cumplen la propiedad `P/1`), `map(+P,?L1,?L2)` (cierto si `P(x_i,y_i)` se cumple para todos los pares `(x_i,y_i)` donde `x_i` e `y_i` son los iésimos elementos de L1 y L2), `include/exclude(+P,+L1,?L2)` (cierto si `L2` está formada por los elementos de la lista L1 que cumplen/no cumplen la propiedad `P/1`), `number/integer/natural(+N)` (cierto si `N` es un número/entero/natural), `?T =.. ?L` (cierto si T = *fun(a1,..,an)* y *L=[fun,a1,..,an]*), `call(+Obj)` (cierto si el objetivo Obj se ejecuta con éxito) y los predicados de recolección de soluciones `bagof(?T, +Obj, ?L)`, `setof(?T, +Obj, ?L)` y `findall(?T, +Obj, ?L)`.

1. (0.5 puntos) Indique qué respuesta(s) ofrecería Prolog ante la siguiente consulta:

```
?- p(E, [_,E|R]) = p(3, [2|[F,4]]), append(A, B, [F|R]),
   length(A, LA), LA =< 1.
```

Solución:

Prolog devolvería dos respuestas:

```
E = F = 3, R = [4], A = [], B = [3, 4], LA = 0
E = F = 3, R = [4], A = [3], B = [4], LA = 1
```

y acabaría con un `false` para indicar que no hay más soluciones.

2. (3 puntos) Proponga una implementación para el predicado `pares(+L, +P, ?N)`, cierto si `N` es la cantidad de elementos situados en posiciones *pares* de la lista `L` que cumplen la propiedad `P/1` , entendiendo que las posiciones de la lista empiezan a contar en 1. Por ejemplo, las consultas `pares([],number,N)`, `pares([1,a,2],number,N)` y `pares([2],number,N)` darían `N=0`, mientras que `pares([1,2,3,a],number,N)` y `pares([1,3],number,N)` darían `N=1`.

Solución:

```
pares([], _, 0).
pares([_], _, 0).
pares([_,D|R], P, N) :-
   Obj =.. [P,D],
   call(Obj),
   !,
   pares(R, P, NR),
   N is NR + 1.
pares([_,_|R], P, N) :-
   pares(R, P, N).
```

3. (1.5 puntos) Dado el predicado

```
mist([C], C) :- !.
mist([C|R], MR) :- mist(R, MR), MR >= C, !.
mist([C|_], C).
```

describa en lenguaje natural (con sus propias palabras) el *resultado* del predicado `mist` (`mist(L,C)` es cierto si ....) y razone si podría producir algún error en tiempo de ejecución y si el corte de la primera cláusula es superfluo o bien su supresión tendría algún efecto.

Solución:

`mist(L,C)` es cierto si L es una lista con C como único elemento o bien L es una lista no vacía de expresiones aritméticas evaluables con máximo C. El predicado numérico `>=` producirá un error en tiempo de ejecución si la lista contiene más de un elemento y alguno de ellos no es numérico. Si se suprimiese el corte de la primera cláusula las consultas con listas de un único elemento y segundo parámetro de salida repetirían la solución (darían solución tanto con la primera como con la tercera cláusula).

4. (3 puntos) Suponga disponibles los siguientes predicados: `asignaturas(?G, ?LA)`, cierto si `LA` es una lista con las asignaturas correspondientes al grado `G`, `usa(?A, ?L)`, cierto si la asignatura `A` usa el lenguaje `L` y `cerrada(?A)`, cierto si la asignatura `A` no se imparte en la actualidad. Teniendo en cuenta lo anterior, implemente el predicado `num_dif(+G, ?N)`, cierto si todos los lenguajes usados actualmente en el grado `G` son distintos entre sí y `N>0` es su número. El predicado debe fallar si en el grado no se usa ningún lenguaje o se usa alguno más de una vez. Por ejemplo, si se dispusiese de los datos `asignaturas(gii,[inf1,inf2,inf3])`,

`asignaturas(gis, [inf1,inf2,inf4])`, `asignaturas(arte, [ar1,ar2])`, `usa(inf1,java)`, `usa(inf2,haskell)`, `usa(inf2,prolog)`, `usa(inf3,java)`, `usa(inf4,python)` y `cerrada(inf4)`, entonces las consultas `?- num_dif(gii, X)` y `?- num_dif(arte, X)` fallarían, mientras que `?- num_dif(gis, X)` daría `X=3`.

Solución:

```
num_dif(Grado, N) :-
    asignaturas(Grado, LAs),
    bagof(Leng,
           A^(member(A, LAs),
             \+ cerrada(A),
             usa(A, Leng)),
          ListaLeng),
    diferentes(ListaLeng),
    length(ListaLeng, N).

diferentes([]).
diferentes([C|R]) :-
    \+ member(C,R),
    diferentes(R).
```

5. (2 puntos) Dado el programa descrito en la otra cara de esta hoja y la consulta

```
?- p(X,Y), \+ f(Y).
```

dibuje el Árbol de Resolución correspondiente, marcando la(s) posible(s) rama(s) podada(s), e indique qué respuesta(s) ofrece Prolog y en qué orden lo hace.

```
p(U,V) :- h(U), !, h(V).          |               h(a).
f(b).                             |               h(b) :- !.
```

Solución:

El árbol de Resolución correspondiente (en el examen había que dibujarlo) tiene, recorriéndolo en profundidad, las siguientes ramas: una rama fallo, una rama éxito con solución `X=a, Y=a`, otra rama fallo y dos ramas podadas por cortes.

Por lo tanto, Prolog devuelve una única respuesta, `[X=a, Y=a]`, y a continuación `false`, indicando que no hay más respuestas.

Universidad Rey Juan Carlos

Prueba de Programación Lógica (A) 18-12-2018

Apellidos: ______________________ **Nombre:** ______________

Grado: GII / GII-ONLINE / GII+GADE / GII+GIC / GII+GIS / GII+MAT

- La duración de esta prueba es de 1 hora y 20 minutos.
- No se permite el uso de ningún tipo de material ni dispositivo electrónico.

Se supondrán disponibles los siguientes predicados: `length(?L,?N)` (cierto si N es la longitud de la lista L), `append(?L1,?L2,?L)` (cierto si L es la concatenación de las listas L1 y L2), `member(?E,?L)` (cierto si E pertenece a L), `sumlist(+L,?N)` (cierto si `L` es una lista de números cuya suma es N), `map(+P,+L)` (cierto si `L` es una lista tal que todos sus elementos cumplen la propiedad `P/1`), `map(+P,?L1,?L2)` (cierto si `P(x_i,y_i)` se cumple para todos los pares `(x_i,y_i)` donde `x_i` e `y_i` son los iésimos elementos de L1 y L2), `include/exclude(+P,+L1,?L2)` (cierto si `L2` está formada por los elementos de la lista L1 que cumplen/no cumplen la propiedad `P/1`), `number/integer(+N)` (cierto si `N` es un número/entero), `?T =.. ?L` (cierto si *T = fun(a1,..,an)* y *L=[fun,a1,..,an]*), `call(+Obj)` (cierto si el objetivo Obj se ejecuta con éxito) y los predicados de recolección de soluciones `bagof(?T, +Obj, ?L)`, `setof(?T, +Obj, ?L)` y `findall(?T, +Obj, ?L)`.

1. En este ejercicio se trabajará con predicados lógicos puros, *sin usar la aritmética de Prolog*, dando por hecho que los naturales se representan mediante `0,s(0),s(s(0)),...`

   (a) (1,5 puntos) Proponga un predicado Prolog para implementar la función de Ackermann, que se define matemáticamente como sigue, siendo M y N naturales:

$$f(M,N) = \begin{cases} N+1 & \text{si } M=0 \\ f(M-1,1) & \text{si } M\neq 0 \text{ y } N=0 \\ f(M-1,f(M,N-1)) & \text{si } M\neq 0 \text{ y } N\neq 0 \end{cases}$$

Solución:

```
ack(0,N,s(N)).
ack(s(M),0,A) :-
    ack(M,s(0),A).
ack(s(M),s(N),A) :-
    ack(s(M),N,A1),
    ack(M,A1,A).
```

   (b) (1 punto) Suponga disponibles `par(?N)`, cierto si N es par, y `suma(?X,?Y,?Z)`, cierto si Z es la suma de los naturales X e Y, y considere el predicado

```
a(Z, L) :-
   setof(X, Y^U^(suma(X,Y,Z),X \= 0,((\+ par(X)) ; suma(X,U,Y))), LL),
   exclude(par, LL, P),
   append(L, [_], P).
```

Indique, para cada una de las siguientes consultas, si Prolog produciría un error o daría alguna respuesta (en este último caso diga cuál(es)):

```
?- a(0,[]).
?- a(s(0), A).
?- a(s(s(s(s(0)))), A).
```

Solución:
Prolog devolvería `false` en la primera consulta, `A=[]` en la segunda y `A=[s(0)]` en la última.

2. (2,5 puntos) Implemente el predicado `elim(+L,+P,+N,?NL)`, cierto si `NL` es la lista resultante tras eliminar de la lista `L` el `N`-ésimo elemento que cumple la propiedad `P/1`. El predicado debe devolver `false` si `L` no es una lista, o `L` tiene menos de `N` elementos cumpliendo `P`, o `N` no es un natural no nulo. Por ejemplo, las consultas `elim(hola,number,3,X)`, `elim([],number,1,X)` y `elim([1,a],number,2,[1,a])` darían `false`, mientras que `elim([1,a,1,3],number,2,X)` devolvería `X=[1,a,3]` y `elim([a,2,c,1,5],number,2,X)` daría `X=[a,2,c,5]`.

Solución:

```
elim(L, P, N, NL) :-
        integer(N),
        N >= 1,
        elim_n_p(L, P, N, NL).

% El primer elemento NO cumple P
elim_n_p([C|R], P, N, [C|NR]) :-
        Obj =.. [P, C],
        \+ call(Obj),
        !,
        elim_n_p(R, P, N, NR).

% El primer elemento cumple P y N=1
elim_n_p([_|R], _, 1, R) :-
        !.

% El primer elemento cumple P y N\=1
elim_n_p([C|R], P, N, [C|NR]) :-
        N1 is N-1,
        elim_n_p(R, P, N1, NR).
```

3. (3 puntos) Considere los siguientes predicados y datos de ejemplo:

```
% examenes(?L): cierto si L es una lista de listas [asignatura,dia]
% relacionando asignaturas con el día de su examen
   examenes([[a1,18],[a2,10],[a3,10],[a4,18]]).
% profesores(?A,?LP)
% cierto si LP es la lista de los profesores de la asignatura A
   profesores(a1, [p1,p2]).       profesores(a2, [p1]).
   profesores(a3, [p1,p2,p3]).    profesores(a4, [p4]).
% sin_exam(?LA): cierto si LA es la lista de asignaturas en las que
% no se va a realizar el examen
   sin_exam([a4]).
```

Teniendo en cuenta lo anterior, implemente el predicado `fechas(+Prof, ?LDias)`, cierto si `LDias` es la lista no vacía y ordenada de los días en los que el profesor `Prof` tiene un único examen. El predicado debe fallar (devolver false) si el profesor no tiene ningún examen o si algún día tiene más de uno. Por ejemplo, con los datos anteriores, las consultas `fechas(p4,L)` y `fechas(p1,L)` fallan (`p4` no tiene exámenes y `p1` tiene dos un mismo día), `fechas(p2,L)` devuelve `L=[10,18]` y `fechas(p3,L)` devuelve `L=[10]`.

Solución:

```
fechas(Profe, LDias) :-
       examenes(LExamenes),
       sin_exam(Sin),
       findall(D, (member([A,D], LExamenes),
                   \+ member(A, Sin),
                   profesores(A, LProfes),
                   member(Profe, LProfes)),
               TodosDias),
       diferentes(TodosDias), % ejercicios tema PL-3
       setof(D, member(D,TodosDias), LDias). % los ordena

diferentes([]).
diferentes([C|R]) :-
       \+ member(C,R),
       diferentes(R).
```

4. Dado el programa Prolog descrito a continuación y la consulta `?- g(X,Y), \+ f(X).`
   (a) (0,5 puntos) Explique en lenguaje natural (con sus propias palabras) qué se pretende al plantear la consulta anterior.

   Solución:
   Lo que se pretende es averiguar si existen objetos X e Y relacionados entre sí mediante la relación "g" y tales que X no cumpla la propiedad "f".

(b) (1,5 puntos) Dibuje el Árbol de Resolución e indique qué respuesta(s) ofrece Prolog, en qué orden lo hace, y qué rama(s) resulta(n) podada(s).

```
g(U,V) :- h(U), p(V,U).     |           h(a).
p(U,V) :- h(U), !.          |           h(b) :- !.
f(b).                       |           h(c).
```

Solución:

El árbol de Resolución correspondiente (en el examen había que dibujarlo) tiene, recorriéndolo en profundidad por la izquierda, las siguientes ramas: una rama fallo, una rama éxito con solución asociada `X=a, Y=a`, dos ramas podadas por cortes, una rama fallo y cuatro ramas podadas por cortes.

Por lo tanto, Prolog devuelve una única respuesta, `X=a, Y=a`, y a continuación `false`, indicando que no hay más respuestas.

Universidad Rey Juan Carlos **Prueba de Programación Lógica (V-A)** 14-1-2019

Apellidos: ______________________ **Nombre:** ______________

Grado: GII / GII-ONLINE / GII+GADE / GII+GIC / GII+GIS / GII+MAT

- La duración de esta prueba es de 1 hora y 20 minutos.
- No se permite el uso de ningún tipo de material ni dispositivo electrónico.

Se supondrán disponibles los siguientes predicados: `length(?L,?N)` (cierto si N es la longitud de la lista L), `append(?L1,?L2,?L)` (cierto si L es la concatenación de las listas L1 y L2), `member(?E,?L)` (cierto si E pertenece a L), `sumlist(+L,?N)` (cierto si `L` es una lista de números cuya suma es N), `map(+P,+L)` (cierto si `L` es una lista tal que todos sus elementos cumplen la propiedad `P/1`), `map(+P,?L1,?L2)` (cierto si `P(x_i,y_i)` se cumple para todos los pares `(x_i,y_i)` donde `x_i` e `y_i` son los iésimos elementos de L1 y L2), `include/exclude(+P,+L1,?L2)` (cierto si `L2` está formada por los elementos de la lista L1 que cumplen/no cumplen la propiedad `P/1`), `number/integer(+N)` (cierto si `N` es un número/entero), `var/nonvar(+X)` (cierto si `X` es/no es una variable), `?T =.. ?L` (cierto si $T = fun(a1,..,an)$ y *L=[fun,a1,..,an]*), `call(+Obj)` (cierto si el objetivo `Obj` se ejecuta con éxito) y los predicados de recolección de soluciones `bagof(?T, +Obj, ?L)`, `setof(?T, +Obj, ?L)` y `findall(?T, +Obj, ?L)`.

1. (1 punto) Implemente el predicado Prolog `transforma(+N, ?NS)`, cierto si `N` es un número natural y `NS` es su representación en lógica pura, donde los naturales se representan mediante la constante `0` y los términos compuestos `s(0),s(s(0)),etc.` El predicado debe fallar (devolver `false`) si `N` no es un número natural. Por ejemplo, `transforma(a,X)` o `transforma(-1,X)` deben fallar, mientras que `transforma(0,X)` y `transforma(2,X)` deben devolver `X=0` y `X=s(s(0))`, respectivamente.

Solución:

```
trans(0,0).
trans(N, s(NS)) :-
integer(N),
N >= 1,
N1 is N-1,
trans(N1,NS).
```

2. (1 punto) Considere la consulta descrita a continuación e indique si Prolog produciría algún error o daría alguna respuesta (especificando error(es) o respuesta(s)):

```
?- A=[a,b], B=[1,2|A], setof(P, C^T^append([C|P],T,B), L),
   map(length,L,LL), append(NL, [_], LL), sumlist(NL,S).
```

Solución:

```
A = [a, b],
B = [1, 2, a, b],
L = [[], [2], [2, a], [2, a, b]],
LL = [0, 1, 2, 3],
NL = [0, 1, 2],
S = 3
```

3. (3 puntos) Implemente el predicado `prodNnoM(+N, +M, ?P)`, cierto si `N` y `M` son números naturales mayores o iguales que 1 y 2, respectivamente, y `P` es el producto de todos los naturales no nulos menores o iguales que `N` que NO son múltiplos de `M`. El predicado no debe producir ningún error en tiempo de ejecución y debe fallar si `N` o `M` no cumplen los requisitos pedidos. Algunos ejemplos:

   - `prodNnoM(a,3,P)`, `prodNnoM(-1,3,P)` deben fallar.
   - `prodNnoM(1,3,P)` debe devolver `P=1` como única salida.
   - `prodNnoM(2,3,P)` y `prodNnoM(3,3,P)` deben devolver `P=2` $(2 * 1)$ como única salida, y `prodNnoM(6,3,P)` debe devolver `P=40` $(5 * 4 * 2 * 1)$ como única salida.

Solución:

```
prodNnoM(N,M,P) :-
        integer(N),
        integer(M),
        N >= 1,
        M >= 2,
        prodNno(N,M,P).

% caso base
prodNno(1,_,1).

%N es múltiplo de M (y por lo tanto distinto de 1)
prodNno(N,M,P) :-
        N mod M =:= 0,
        !,
        N1 is N - 1,
        prodNno(N1,M,P).

%N NO es múltiplo de M y distinto de 1
prodNno(N,M,P) :-
```

```
        N > 1,
        N1 is N - 1,
        prodNno(N1,M,P1),
        P is N*P1.
```

4. (2,5 puntos) Considere el ejemplo estudiado en clase relativo al *mundo de los bloques*, y suponga disponibles los predicados `encima(X,Y)`, cierto si el bloque `X` está colocado justo encima del bloque `Y`, y `apilado(X,Y)`, cierto si el bloque `X` está apilado sobre el bloque `Y`. Implemente el predicado `bloques(?L)`, cierto si L es una lista conteniendo todos los pares `(B,LB)` donde `B` es un bloque sobre el que hay apilados al menos otros dos bloques (excluyendo al que está justo encima suyo) y `LB` es una lista conteniendo los bloques apilados sobre `B` (excluyendo al que está justo encima suyo). El predicado debe devolver la lista vacía si no hubiese ningún bloque con las características requeridas. Algunos ejemplos: (1) si se tuviese la pila de bloques `abc`, con `c` en la cima, `bloques(L)` devolvería `L=[]` ; (2) con la pila `abcd`, con `d` en la cima, se tendría `L=[(a,[c,d])]` ; (3) con la pila `abcde`, con `e` en la cima, se tendría `L=[(a,[c,d,e]), (b, [d,e])]`.

Solución:

```
bloques(L) :-
    findall(
      (B,LApilados),
      (bagof(A, (apilado(A,B), \+ encima(A,B)), LApilados),
       length(LApilados, CuantosApilados),
       CuantosApilados >= 2),
      L).
```

5. Considere el programa Prolog descrito en la otra cara de esta hoja.
   (a) (1 punto) Explique en lenguaje natural (con sus propias palabras) qué devuelve el predicado `q(L,P,X)` : `q(L,P,X)` es cierto si ....

   Solución:
   `q(L,P,X)` devuelve cierto si X es el primer elemento de la lista L que no cumple la propiedad P/1.

   (b) (1,5 puntos) Dada la consulta

   `?- q([a,2,3], atom, E), \+ r(E).`

   (recuerde que `atom(+X)` es cierto si X es una constante no numérica), dibuje el Árbol de Resolución correspondiente e indique qué respuesta(s) ofrece Prolog, en qué orden lo hace, y qué rama(s) resulta(n) podada(s).

```
q([X|Xs], P, Y) :- O =.. [P, X], call(O), !, q(Xs, P, Y).
q([X|_], _, X).
r(X) :- X mod 2 =:= 1.
```

Solución:

El árbol de Resolución correspondiente (en el examen había que dibujarlo) tiene, recorriéndolo en profundidad por la izquierda, las siguientes ramas: dos ramas fallo, una rama éxito con solución asociada `E=2` y una rama podada por un corte.

Por lo tanto, Prolog devuelve una única respuesta, `E=2`, y a continuación `false`, indicando que no hay más respuestas.