NOMBRE:

PROGRAMACIÓN DECLARATIVA CURSO 2017-18 EXAMEN FINAL 23-1-2018

• Cada pregunta tiene (espero) una y solo una respuesta correcta. Marcad con un aspa la opción elegida.
• **Cada respuesta correcta suma un punto; cada respuesta incorrecta resta medio punto**; las respuestas en blanco ni suman ni restan. Estad ojo avizor y suerte. Está prohibidísimo copiar.

1. Considérense las expresiones, en las que los numerales `2, 1` se suponen de tipo `Int`:
`([2],1)` `(2:[],1:[])` `(2:[]):1` `[2]:[1]:[]`
¿Cuál de las siguientes afirmaciones es cierta?
◯ Hay exactamente tres que están mal tipadas
◯ Hay exactamente dos que están mal tipadas
⊗ Hay exactamente una que está mal tipada

---

2. Considérense las expresiones de tipo (que solo difieren en los paréntesis): τ_1 = `(a -> a -> a) -> a -> (a -> a)`
τ_2 = `(a -> (a -> a)) -> a -> a -> a`
τ_3 = `a -> (a -> a) -> a -> a -> a`
◯ $\tau_1 \equiv \tau_2 \equiv \tau_3$
⊗ $\tau_1 \equiv \tau_2 \not\equiv \tau_3$
◯ $\tau_1 \not\equiv \tau_2 \not\equiv \tau_3 \not\equiv \tau_1$

---

3. Considérese el operador `infixl 4 **` y las expresiones (que solo difieren en los paréntesis): e_1 = `f x y ** y ** x`
e_2 = `(**) (f x y) (** y x)`
e_3 = `(** x) ((** y) ((f x) y))`
⊗ $e_1 \equiv e_3 \not\equiv e_2$
◯ $e_1 \not\equiv e_2 \not\equiv e_3 \not\equiv e_1$
◯ $e_1 \equiv e_2 \not\equiv e_3$

---

4. La evaluación de la expresión
`foldr (\x y -> not y)` e `[False,True,undefined]` da como resultado
⊗ `True`, para **alguna expresión** e de tipo `Bool`
◯ Un error, para **toda expresión** e de tipo `Bool`
◯ Las dos anteriores son falsas.

---

5. La evaluación de la expresión
`foldl (\x y -> not y)` e `[False,True,undefined]` da como resultado
◯ `True`, para **alguna expresión** e de tipo `Bool`
⊗ Un error, para **toda expresión** e de tipo `Bool`
◯ Las dos anteriores son falsas.

---

6. La evaluación de `[take j [3..i] | i <- [1..4], i > 2, j <- [i-1,i]]` produce como resultado
◯ Una lista de números, siendo los dos primeros iguales entre sí
◯ Una lista de listas de números, siendo las dos primeras listas vacías
⊗ Una lista de longitud cuatro, cuyos dos últimos elementos son iguales

---

7. La reducción de la expresión `(\x y z -> x y (y z)) (\x y -> x y) (\x -> x+1) 2` producirá el resultado
◯ 3 ⊗ 4 ◯ 5

---

8. Sea `f` definida por `f x y z = x y (y z)`. El tipo de `f` es:
⊗ `((a -> b) -> b -> c) -> (a -> b) -> a -> c`
◯ `(b -> a -> c)-> (b -> a) -> b -> c`
◯ `f` está mal tipada

---

9. La unificación de `[X|[X,U|Y]]` con `[b,Z,Z]`
◯ No tiene éxito
⊗ Tiene éxito, con las ligaduras `X/b,Z/b,U/b,Y/[]`
◯ Tiene éxito, con las ligaduras `X/b,Z/b,U/[],Y/[b]`

---

10. Sea e una expresión de un cierto tipo. Considérense las siguientes situaciones:
- La evaluación de `length e` termina pero la de `head e` no
- La evaluación de `length e` termina pero la de `last e` no
- `last e` da error de tipos y la evaluación de `length e` termina

Se tiene que:
◯ Ninguna de las tres situaciones es posible
◯ Una de las tres situaciones es posible pero las otras dos no
⊗ Dos de las situaciones son posibles pero la otra no

---

11. Sea `f` definida por las siguientes ecuaciones:
```
f  y  False  = not y
f  x  y      = x && y
```
¿Cuál de las siguientes afirmaciones es cierta?
◯ La función no es estricta en ninguno de sus dos argumentos
◯ La función es estricta en el segundo pero no en el primer argumento
⊗ Las dos anteriores son falsas.

---

12. ¿Cuál de las siguientes funciones f hace que la expresión `map` f `(iterate (takeWhile (< 10)) (iterate (+ 1) 0))` esté **mal tipada**? *(Suponemos que* $0, 1, 10$ *son de tipo Int)*
◯ $f \equiv$ `take 5`
◯ $f \equiv$ `length`
⊗ $f \equiv$ `(+ 1)`

---

13. ¿Qué podemos afirmar de la evaluación de `last (filter (<` n`) (iterate (+ 1)` m`))` , siendo n y m dos números concretos de tipo `Int`?
◯ La evaluación terminará, con independencia de n y m
⊗ La evaluación no terminará, con independencia de n y m
◯ Las dos anteriores son falsas.

---

14. ¿ Cuál de los siguientes programas lógicos expresa de forma natural la función Haskell dada por
```
f(Z) = S Z
f(S x) = S (f x)
```
?

◯
```
f(z,s(z)).
f(s(X),Y) :- f(X,s(Y)).
```

⊗
```
f(z,s(z)).
f(s(X),s(Y)) :- f(X,Y).
```

◯
```
f(z,s(z)).
f(s(X),s(f(X,Y))).
```

---

15. La evaluación de la expresión `let {y= 1:x ; x=y++[2]} in head x` produce como resultado:
⊗ `1`
◯ Un error
◯ Un cómputo no terminante

---

16. ¿Cuántas de las siguientes definiciones de tipos (independientes unas de otras) son correctas?
```
data Tip  = A  | B Int Tip      | C (Int,Tip,Tip)
data Tap  = A  | B (Int,Bool)   | A (Int,Int,Tap)
data Top  = A  | B a
```
⊗ Una de las tres
◯ Dos de las tres
◯ Ninguna de las tres

---

17. ¿Cuáles de las siguientes expresiones representan correctamente la acción de IO que lee una línea y escribe su longitud?
```
let x = getLine in length x
```
```
return (length getLine)
```
```
do  x <- getLine
    print (length x)
```
◯ La segunda y la tercera
◯ La segunda y ninguna otra
⊗ Las dos anteriores son falsas.

---

18. El objetivo Prolog `var(Y),f(0,1,Y) =.. [F,U,V,2],Y is U+V`
◯ Tiene éxito y `Y` queda ligada a `1` (y puede haber más ligaduras)
◯ Tiene éxito y `Y` queda ligada a `2` (y puede haber más ligaduras)
⊗ No tiene éxito

PROGRAMACIÓN DECLARATIVA CURSO 2017-18 23 de enero de 2018

Nota previa: debe declararse el tipo de todas las funciones que se programen

1. **(1 punto)**

   (a) Escribe una expresión Haskell cuya evaluación produzca la lista infinita
   `[(1,1),(4,2),(3,9),(16,4),(5,25),(36,6),...`
   Nota: puedes usar funciones del preludio de Haskell, listas intesionales o lambda expresiones, pero no otras funciones auxiliares.

   (b) Razona brevemente cuál es el tipo de la función definida por la ecuación
   `f x y z  = x y (y z)`

2. **(2 puntos)**

   - Define un tipo de datos polimórfico para representar árboles generales, en los que cada nodo tiene una información y n hijos ($n \geq 0$, y puede variar con cada nodo). No se consideran árboles vacíos.
   - Programa las siguientes funciones:
     - `suma t`, que obtiene la suma de los contenidos de todos los nodos del árbol `t`.
     - `creciente t`, que es la propiedad que expresa que para cada nodo del árbol `t` y cada hijo `t'` de ese nodo, la suma de los nodos de `t'` es mayor que la suma de todos los nodos de todos los hermanos de `t'` situados a su izquierda.
   - Declara explícitamente el tipo de los árboles como instancia de la clase `Eq`, de manera que el orden definido sea el mismo que resultaría de usar `deriving Eq`.

3. **(1 punto)**
   Define (sin utilizar `foldr`) la siguiente variante de `foldr` que opera con listas **no vacías**, especificada como:
   `foldr1` $\bigoplus [x_1, x_2, \ldots, x_n] = x_1 \bigoplus (x_2 \bigoplus (x_3 \ldots \bigoplus (x_{n-1} \bigoplus x_n) \ldots))$
   Expresa mediante `foldr1` la función `last`.

4. **(1 punto)**
   Dado el programa lógico

```
p(a).          q(c(X,Y),Z) :- q(X,Z).
p(b).          q(c(X,Y),Y).
```

   (i) Construye el árbol de resolución del objetivo `q(c(c(c(b,d),Y),a),X),p(X).`

   (ii) Indica cómo cambia el árbol si la primera cláusula de `p` se cambia por `p(a) :- !.`

   (ii) Indica cómo cambia el árbol si la primera cláusula de `q` se cambia por `q(X,c(Y,Z)) :- q(X,Z),!.`
   (Este cambio es independiente del anterior)

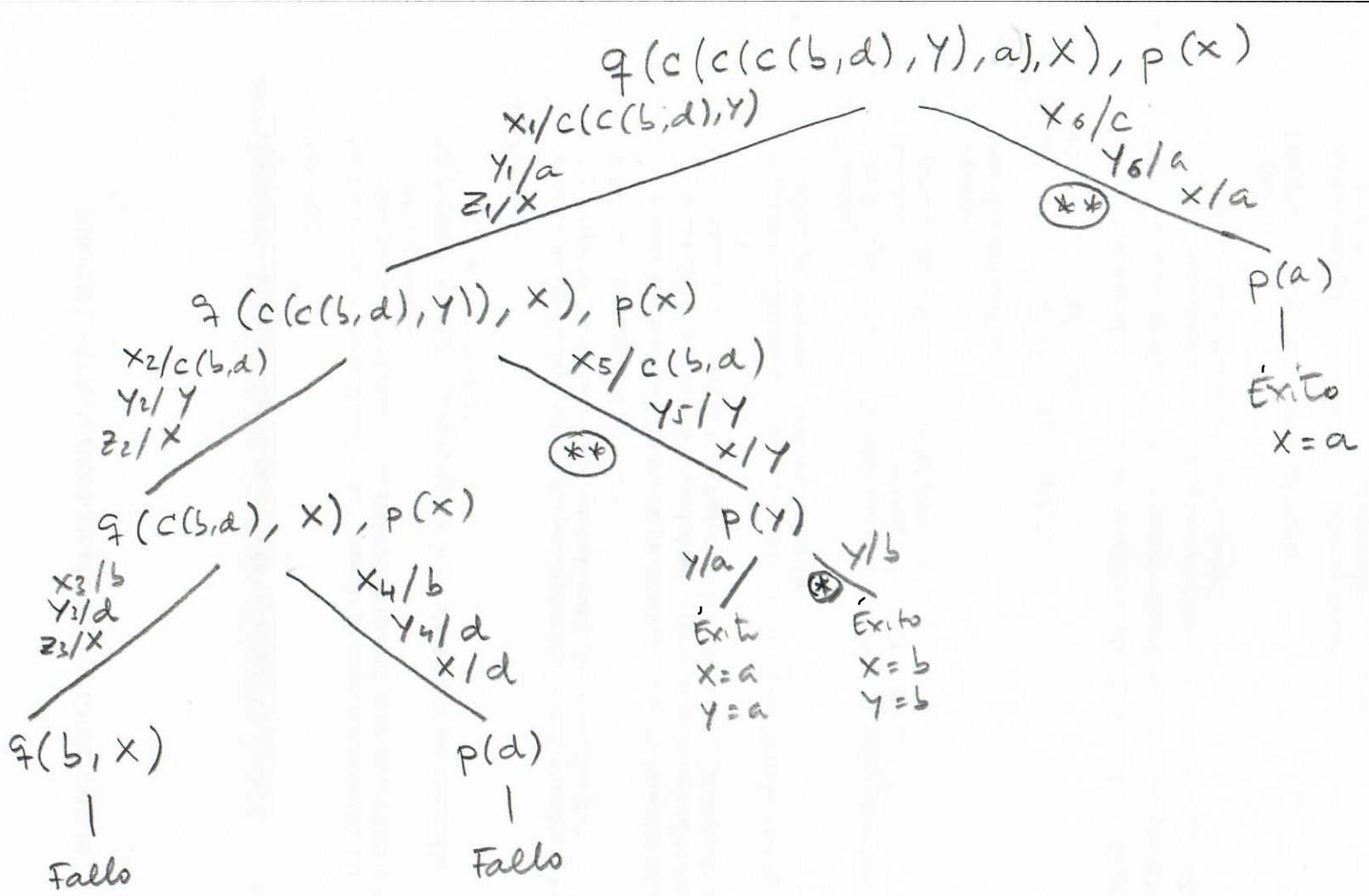
5. **(1 punto)**
   Programa en Prolog los siguientes predicados:

   (a) `mas_corta(Xs,Ys)` $\Leftrightarrow$ la lista `Xs` tiene menos longitud que `Ys`.
   Nota: no pueden usarse predicados primitivos, ni siquiera `is`.

   (b) `rep_sum(Xs,Ys)` $\Leftrightarrow$ `Ys` resulta de reemplazar cada elemento de `Xs` por la suma de todos los elementos de `Xs`.
   Ejemplo: `rep_sum([1,2,3,1],Ys)` debe tener éxito con respuesta `Ys = [7,7,7,7]`.
   Nota: se supone, sin necesidad de comprobación, que `Xs` *es una lista formada por números.*
   Renota: programarlo con un solo recorrido de `Xs` *tiene una bonificación de 0,5 puntos.*

```
--------- Examen de Programación Declarativa, 23 de enero de 2018 -------
--------------------- Esbozo de solución -----------------

-- Ejercicio 1a

e = [if mod i 2 == 1 then (i,i^2) else (i^2,i)|i <- [1..]]

-- Ejercicio 1b

f x y z = x y (y z)

Sean tf, tx, ty, tz los tipos de f, x, y , z.
Se tiene
tf = tx -> ty -> tz -> t
ty = tz -> t', siendo entonces t' el tipo de (y z)
tx = ty -> t' -> t = (tz -> t') -> t' -> t
Luego
tf = ((tz -> t') -> t' -> t) -> (tz -> t') -> tz -> t
O bien, renombrando variables,
tf = ((a -> b) -> b -> c) -> (a -> b) -> a -> c


-- Ejercicio 2
-- Un árbol viene dado por un nodo con un contenido y una lista de hijos, que
son también árboles.
-- Si la lista es vacía, se tratará de una hoja

data Arbol a = Nodo a [Arbol a]

suma:: Num a => Arbol a -> a
suma (Nodo x hijos) = x + sum (map suma hijos)

creciente:: (Num a,Ord a) => Arbol a -> Bool

-- Una versión muy simple, aunque algo a la brava y no muy eficiente
--    Para ver si un árbol es creciente
--     comprobamos que cada hijo de la raíz suma más que todos los hermanos a
su izquierda juntos
--     y comprobamos recursivamente que cada hijo de la raíz es además
creciente

creciente (Nodo _ hijos) =
  and [suma (hijos!!i) > sum [suma (hijos!!j) | j <- [0..i-1]]|i <- [0 ..
length hijos-1]]
  &&
  and (map creciente hijos)

instance Eq a => Eq (Arbol a) where
  Nodo x hs == Nodo x' hs' = x==x' && hs==hs'


-- Ejercicio 3
foldr1:: (a->a->a) -> [a] -> a
foldr1 f [x]    = x
foldr1 f (x:xs) = f x (foldr1 f xs)

last:: [a] -> a
last = foldr1 (\x y -> y)
```

```
-- Nota: para probar este código en el intérprete conviene dar otro nombre
--   tanto a foldr1 como a last, pues ambas funciones existen en Prelude

--  Ejercicio 4
(i) Ver árbol en dibujo adjunto
(ii) Se poda la rama marcada con (*)
(iii) Se podan las ramas marcadas con (**), por lo que no quedan nodos de éxito

--  Ejercicio 5a
mas_corta([],_). % Aunque si interpretamos 'menos longitud' en sentido
estricto, seria mas_corta([],[_|_]).
mas_corta([_|Xs],[_|Ys]) :- mas_corta(Xs,Ys).

--  Ejercicio 5b
%% Una solución muy directa: calculo la suma S de todos los elementos de Xs y
luego reemplazo cada elemento por S
rep_sum(Xs,Ys) :-
  suma(Xs,S),
  reemplaza(Xs,S,Ys).

suma([],0).
suma([X|Xs],S) :-
  suma(Xs,S1),
  S is X+S1.

reemplaza([],_,[]).
reemplaza([X|Xs],Y,[Y|Ys]) :- reemplaza(Xs,Y,Ys).

%% Una solución con un solo recorrido, gracias al poder de la variable lógica
%% Según recorremos Xs vamos calculando la suma acumulada y reemplazando
%% cada elemento de Xs por una misma variable, que al final se instancia a la
suma calculada.

rep_sum(Xs,Ys) :- rep_sum_aux(Xs,Y,0,Ys).

rep_sum_aux([],S,S,[]).
rep_sum_aux([X|Xs],Y,S,[Y|Ys]) :-
  S1 is X+S,
  rep_sum_aux(Xs,Y,S1,Ys).
```


q(c(c(c(b,d),Y),a),X), p(X)
X1/c(c(b,d),Y)
Y1/a
Z1/X
X6/c
Y6/a
X/a
(**)
p(a)
Éxito
X = a
q(c(c(b,d),Y)),X), p(X)
X2/c(b,d)
Y2/Y
Z2/X
X5/c(b,d)
Y5/Y
X/Y
(**)
q(c(b,d),X), p(X)
p(Y)
X3/b
Y3/d
Z3/X
X4/b
Y4/d
X/d
Y/a
Y/b
(*)
Éxito
X = a
Y = a
Éxito
X = b
Y = b
q(b,X)
p(d)
Fallo
Fallo