18 EJERCICIOS DE EXÁMENES

18.1. AÑO 1996

18.1 (PD: Febrero,96) Utilizando únicamente la función estándar

$$\begin{array}{l} foldr :: (a \to b \to b) \to b \to [a] \to b \\ foldr \_\ z\ [\,] \quad\ = z \\ foldr\ f\ z\ (x : xs) = f\ x\ (foldr\ f\ z\ xs) \end{array}$$

(A).– Describe la función

$$h\ p\ xs\ =\ [\ x \mid x \leftarrow\ xs,\ p\ x\ ]$$

(B).– Escribe una función para encontrar los divisores de un entero

$$divisores\ ::\ Int\ \to\ [Int]$$

(C).– Si representamos el polinomio $\sum_{k=0}^{n} a_k x^k$ con la lista $[a_0, \ldots,\ a_n]$, escribe una función para encontrar la mayor raíz entera de un polinomio de coeficientes enteros

$$\begin{array}{l} \textbf{data}\ Raíz\ a\ =\ NoExiste \mid Raíz\ a\ \textbf{deriving}\ Show \\ mayorRaízEntera\ ::\ [Int]\ \to\ Raíz\ Int \end{array}$$

AYUDA.– Toda raíz entera de un polinomio de coeficientes enteros es un divisor del término independiente.

18.2 (PD: Febrero,96) Pon un ejemplo de función polimórfica y uno de función sobrecargada que admitan por tipo $a \to a$.

18.3 (PD: Febrero,96) **(A)**.– Siendo $x :: \beta$, prueba que entonces

$$(\lambda\ g\ \to\ g\ x) :: (\ \beta\ \to\ \delta) \to\ \delta$$

(B).– Utilizando lo anterior, infiere el tipo de la función pen que verifica la ecuación

$$pen\ f\ x\ =\ f\ (\lambda g\ \to\ g\ x)$$

18.4 (PD: Febrero,96) Sean las declaraciones para la diferencias de enteros

$$\begin{array}{l}\textbf{data}\ Ent\ =\ O \mid S\ Ent \mid P\ Ent\ \textbf{deriving}\ Show\\ \textbf{instance}\ Num\ Ent\ \textbf{where}\\ \quad x - O\ \ \ \ \ \ = x\\ \quad x - (S\ y)\ = P(x-y)\\ \quad x - (P\ y)\ = S(x-y)\end{array}$$

Demuestra, por inducción estructural, $\forall\ x\ \centerdot\ x :: Ent\ \centerdot\ O - (O - x) = x$

18.5 (PD: Junio,96) Demuestra la igualdad siguiente

$$f.head\ =\ head.map\ f$$

planteando claramente los pasos de la demostración.

18.6 (PDI: Junio,96, y PD: Setiembre, 96)
(A).– Define una estructura $Ater\ a$ de árbol ternario (árboles vacíos o con tres ramas exactamente) con información en los nodos.
(B).– Define, para la estructura anterior, una función $nNodos$ que produzca el número de nodos de un árbol ternario.
(C).– Define una función de plegado para árboles ternarios con el siguiente tipo

$$pliegat\ ::\ (a\ \to\ b\ \to\ b\ \to\ b\ \to\ b)\ \to\ b\ \to\ Ater\ a\ \to\ b$$

(D).– Redefine la función $nNodos$ utilizando la función de plegado anterior.

18.7 (PDI: Junio,96) Dada la sucesión siguiente

$$[[(1,1)], [(2,1),(1,2)], [(3,1),(2,2),(1,3)],\ \ldots]$$

(A).– Encuentra una expresión funcional para dicha sucesión.
(B).– Describe la sucesión como una red de procesos.

18.8 (LabIV: Junio,96, y LabIV: Febrero, 98)
(A).– Escribe funciones

$$\begin{array}{ll}suma & :: Int\ \to\ Int\ \to\ Int\ \to\ (Int, Int)\\ test & :: Int\ \to\ Int\ \to\ Int\ \to\ (Int, Int)\ \to\ Bool\end{array}$$

donde $suma$ calcula la suma de tres dígitos devolviendo en forma de par el *acarreo* y la suma módulo 10, y $test$ comprueba que la función anterior es correcta; por ejemplo:

$$\begin{array}{ll}suma\ 1\ 8\ 7 & \Longrightarrow\ (1,\ 6)\\ test\ 1\ 8\ 7\ (1,6) & \Longrightarrow\ True\\ test\ 1\ 8\ 7\ (1,5) & \Longrightarrow\ False\end{array}$$

(B).– Escribe una función

$$sumaListas :: [Int] \rightarrow [Int] \rightarrow [Int]$$

para calcular en forma de lista la suma de dos números dados por las listas de sus cifras en base 10, en el supuesto de que las dos listas argumentos sean de la misma longitud; por ejemplo:

```
MAIN> sumaListas [9,2,4,5,6] [3,9,9,3,7]
[1,3,2,3,9,3]
```

(C).– Se pretende resolver el siguiente puzzle: asignar dígitos a las letras en el siguiente cuadro para que la suma sea correcta:

$$\begin{array}{cccc} & a & n & a \\ + & a & m & a \\ \hline & m & a & s \end{array}$$

Para ello damos el siguiente esquema de solución basado en listas por comprensión:

```
ds  = [1..9]
sol = [ [(a,n,a),(a,m,a),(m,a,s)] |
        a ← ds,
        let (ac,s) = suma 0 a a,
        n ← ds,
        m ← ds,
        let (ac',a') = suma ac n m, a== a',
        ... ]
```

Completa la lista por comprensión anterior.

(D).– Modifica el apartado (C) si no se permiten dígitos repetidos ¿Qué conjunto de dígitos ds proporciona una única solución?

18.9 (LabIV: Junio,96) **(A)**.– Escribe funciones

$$resta :: Int \rightarrow Int \rightarrow Int \rightarrow (Int, Int)$$
$$test :: Int \rightarrow Int \rightarrow Int \rightarrow (Int, Int) \rightarrow Bool$$

donde *resta* calcula la diferencia entre dos dígitos (segundo y tercer argumentos) con un posible *acarreo* anterior (primer argumento), devolviendo en forma de par el nuevo acarreo y la diferencia módulo 10; *test* comprueba que la función anterior es correcta; por ejemplo:

```
MAIN> resta 1 8 3
(1,4)   -- de 9(8+1) a 13 van 4 y me llevo 1

MAIN> resta 1 3 8
(0,4)   -- de 4(3+1) a 8 van 4 y me llevo 0

MAIN> test 1 8 3 (1,4)
True
```

$$\begin{array}{l}\text{MAIN}> \ test\ 1\ 8\ 3\ (0,5)\\ False\end{array}$$

(B).– Escribe las funciones

$$\begin{array}{lll} mayor & :: & [Int] \rightarrow [Int] \rightarrow Bool \\ restaListas & :: & [Int] \rightarrow [Int] \rightarrow [Int] \end{array}$$

donde los argumentos son las listas de los dígitos (base 10) correspondientes a dos números (se consideran las listas de igual longitud); $mayor$ comprueba si la lista segundo argumento corresponde a un número mayor que el correspondiente a la lista primer argumento; $restaListas$ calcula, en forma de lista, la diferencia de los números representados por las listas argumentos en el supuesto de que ambas sean de la misma longitud y la primera represente un número menor que la segunda; por ejemplo:

$$\begin{array}{l} restaListas\ [3,9,9,3,7]\ [9,2,4,5,6] \\ \Longrightarrow \\ [5,2,5,1,9] \end{array}$$

(C).– Se pretende resolver el siguiente puzzle: asignar dígitos a las letras del siguiente cuadro para que la diferencia sea correcta:

$$\begin{array}{cccc} & m & a & s \\ - & a & n & a \\ \hline & a & m & a \end{array}$$

Para ello damos el siguiente esquema de solución basado en listas por comprensión:

$$\begin{array}{l} ds\ =\ [1..9] \\ solr\ =[\ [(m,a,s),(a,n,a),(a,m,a)]\ | \\ \qquad\qquad s \leftarrow\ ds, \\ \qquad\qquad a \leftarrow\ ds, \\ \qquad\qquad \mathbf{let}\ (ac,a') = resta\ 0\ a\ s,\ a'\texttt{==}\ a, \\ \qquad\qquad n \leftarrow\ ds, \\ \qquad\qquad \mathbf{let}\ (ac',m)\ =\ resta\ ac\ n\ a, \\ \qquad\qquad \ldots\ ] \end{array}$$

Completa la lista por compresión anterior.

(D).– Modifica el apartado (C) para el supuesto en que no se permitan dígitos repetidos ¿Qué conjunto ds de dígitos proporciona en este caso una solución única?

18.10 (PD: Setiembre,96) **(A)**.– ¿Cuál es el tipo de la función g definida en la forma $g\ =\ map\ f\ .\ map\ f$?
(B).– Demuestra que $map\ f\ .\ map\ f\ =\ map\ (f.f)$

18.11 (LabIV: Setiembre,96) **(A)**.– Escribe una función

$$estáYresto :: Eq\ a \Rightarrow a \rightarrow [a] \rightarrow (Bool, [a])$$

tal que $estáYresto\ x\ xs \Longrightarrow (e, rs)$, donde el primer elemento del par que devuelve es el resultado del test x ***pertenece a*** xs y el segundo la lista xs sin la primera aparición del elemento (o la propia lista si no aparece).
(B).– Escribe una función

$$contenidoYresto :: Eq\ a \Rightarrow [a] \rightarrow [a] \rightarrow (Bool, [a])$$

tal que $contenidoYresto\ x\ xs \Longrightarrow (e, rs)$, donde el primer elemento del par que devuelve es el resultado del test xs ***está contenido en*** ys (considerando xs e ys como multiconjuntos) y cuyo segundo elemento es la lista diferencia $ys \backslash\backslash\ xs$:

MAIN> $contenidoYresto\ [1,2,2]\ [3,2,1,1]$
$(False,\ [\,])$

MAIN> $contenidoYresto\ [1,2,2]\ [3,2,2,1]$
$(True,\ [3])$

18.12 (PDI: Setiembre,96) **(A)**.– Justifica qué calcula la siguiente función

$$\begin{aligned} h\ [\,] &= [\,[\,]\,] \\ h\ (x : xs) &= concat\ [\,[x : t, t] \mid t \leftarrow h\ xs\,] \end{aligned}$$

(B).– Escribe las funciones

$divisores :: Int \rightarrow [Int]$
– – calcula los divisores naturales de un número natural

$perfecto :: Int \rightarrow Bool$
– – comprueba si un número natural es perfecto; o sea, es
– – la suma de sus divisores propios

18.13 (PDI: Setiembre,96) **(A)**.– Sean las declaraciones para una posible representación de los enteros, la suma y diferencia de éstos

$$\textbf{data}\ Ent = O \mid S\ Ent \mid P\ Ent\ \textbf{deriving}\ Show$$

$$\textbf{instance}\ Num\ Ent\ \textbf{where}$$

$$\begin{aligned} O + y &= y \\ (S\ x) + y &= S\ (x + y) \\ (P\ x) + y &= P\ (x + y) \end{aligned}$$

$$\begin{aligned} x - O &= x \\ x - (S\ y) &= P(x - y) \\ x - (P\ y) &= S(x - y) \end{aligned}$$

Demuestra por inducción estructural, $\forall\ x\ .\ x :: Ent\ .\ O - (O - x) = x$.
(B).– Utilizando únicamente la función estándar

$$\begin{array}{l} foldr \quad :: (a \to b \to b) \to b \to [a] \to b \\ foldr\ \_\ z\ [\,] \quad = z \\ foldr\ f\ z\ (x : xs) \ = f\ x\ (foldr\ f\ z\ xs) \end{array}$$

describe las funciones:

$$suma :: [Ent] \to Ent$$
– – la suma de los elementos de la lista de enteros

$$pertenece :: Ent \to [Ent] \to Bool$$
– – el test de pertenencia

(C).– Escribe funciones de conversión entre los tipos Ent e Int

$$\begin{array}{l} aEnt :: Int \to Ent \\ aInt :: Ent \to Int \end{array}$$

de tal forma que, por ejemplo:

$$\begin{array}{l} \text{MAIN}> \ aEnt\ 3 \\ S\ (S\ (S\ O)) \end{array}$$

$$\begin{array}{l} \text{MAIN}> \ aInt\ (P(S(P\ O))) \\ -1 \end{array}$$

Utiliza las funciones anteriores para definir una función

$$máximo :: Ent \to Ent \to Ent$$

18.14 (PDI: Setiembre,96) Escribe una red de procesos que describa una función $lista :: [\,[Int]\,]$ que genere la lista infinita de listas infinitas

$$[\,[1,2,3,4,\ldots],\ [1,4,9,16,\ldots],[1,8,27,64,\ldots],\ \ldots]$$

y aplica la red anterior para calcular la lista infinita

$$sol \implies [1^2,\ 2^3,\ 3^4,\ \ldots]$$

18.15 (PDI: Setiembre,96) **(A)**.– Escribe una función

$$comunes :: Eq\ a \ \Rightarrow [a] \to [a] \to [a]$$

para devolver la lista de los elementos comunes de sus listas argumentos
(B).– Escribe una función con tipo

$$pasa \;::\; Eq\ a \;\Rightarrow\; Int \;\to\; [a] \;\to\; [a] \;\to\; ([a],[a])$$

para *pasar* a la lista segundo argumento tantos elementos (no pertenecientes a la primera lista) como indique el primer argumento extrayendo desde la lista tercer argumento y devolviendo también la lista restante. Por ejemplo

$$\begin{array}{l} pasa\ 2\ [1,2]\ [1,3,5,8,4] \;\Longrightarrow ([5,3,1,2],[8,4]) \\ pasa\ 1\ [1,2]\ [1,2,3,5,8,4] \;\Longrightarrow ([3,1,2],[5,8,4]) \end{array}$$

(C).– La función *incluye* incrementa en uno el contador que ocupa la posición indicada por el primer argumento en la lista de contadores dada por el segundo argumento; por ejemplo

$$\begin{array}{l} incluye\ 0\ [10,12,13,15,20,12,20,0] \;\Longrightarrow [11,12,13,15,20,12,20,0] \\ incluye\ 3\ [10,12,13,15,20,12,20,0] \;\Longrightarrow [10,12,13,16,20,12,20,0] \end{array}$$

18.16 (LabIV: Noviembre, 96; PD: Febrero, 97) Sea la siguiente estructura

$$\mathbf{data}\ Cola\ a \;=\; Ult\ a \mid (Cola\ a) \;:>\; a\ \mathbf{deriving}\ Show$$

para representar estructuras con comportamiento de *colas* de tipo base a. Por ejemplo, en la cola $Ult\ 3 \;:>\; 2) \;:>\; 1$ el último elemento es el 3. Las operaciones de inserción y borrado serían:

$$\begin{array}{l} \text{MAIN}> \ enCola\ 4\ ((Ult\ 3 \;:>\; 2) \;:>\; 1\ ) \\ ((Ult\ 4 \;:>\; 3) \;:>\; 2) \;:>\; 1 \;::\; Cola\ Integer \end{array}$$

$$\begin{array}{l} \text{MAIN}> \ deCola\ ((Ult\ 3 \;:>\; 2) \;:>\; 1\ ) \\ (\ Ult\ 3 \;:>\; 2\ ,\ 1\ ) \;::\; (Cola\ Integer,\ Integer) \end{array}$$

(A).– Define las funciones de inserción y borrado

$$\begin{array}{l} enCola \;::\; a \;\to\; Cola\ a \;\to\; Cola\ a \\ deCola \;::\; Cola\ a \;\to\; (Cola\ a,\ a) \end{array}$$

Define la función $mapCola$ que aplica una función a todos los elementos de una cola

$$mapCola \;::\; (a \;\to\; b) \;\to\; Cola\ a \;\to\; Cola\ b$$

y demuestra por inducción estructural que, $\forall\, f,\ x,\ c \;\;\centerdot\; f :: a \to b,\ \ x :: a$, $c :: Cola\ a$:

$$mapCola\ f\ (enCola\ x\ c) \;=\; enCola\ (f\ x)\ (mapCola\ f\ c)$$

(B).– Sea ahora la siguiente función de reducción de colas, donde el segundo argumento es una operación a realizar cuando la cola tenga un solo elemento

$$
\begin{array}{l}
redCola \qquad\qquad :: (a \to b \to b) \to (a \to b) \to Cola\ a \to b \\
redCola\ \_\ g\ (Ult\ x) \quad = g\ x \\
redCola\ f\ g\ (c :> y) \ = f\ y\ (redCola\ f\ g\ c)
\end{array}
$$

Utilizando el esquema de reducción anterior, define las funciones $enCola$ (añade a una cola), $colaALista$ (transforma una cola en una lista), de forma que:

$$
\begin{array}{l}
colaALista\ (\ (Ult\ 3 :> 2) :> 1\ ) \\
\Longrightarrow \\
[1,2,3]
\end{array}
$$

(C).– Define también funciones que inviertan colas, fusionen dos colas, y comprueben si un objeto aparece en una cola

$$
\begin{array}{l}
inversa \quad :: Cola\ a \to Cola\ a \\
fusión \quad :: Cola\ a \to Cola\ a \to Cola\ a \\
pertenece :: Eq\ a \Rightarrow a \to Cola\ a \to Bool
\end{array}
$$

de forma que

$$
\begin{array}{l}
\text{MAIN}> \ inversa\ (Ult\ 3 :> 2) :> 1 \\
(Ult\ 1 :> 2) :> 3
\end{array}
$$

$$
\begin{array}{l}
\text{MAIN}> \ fusión\ ((Ult\ 3 :> 2) :> 1\ )\ (\ Ult\ 5 :> 4) \\
(((Ult\ 3 :> 2) :> 1) :> 5) :> 4
\end{array}
$$

Usando **infix**, define $:>$ para que sea asociativo a la izquierda y así eliminar paréntesis.
(D).– Define, utilizando *foldl*, una función para pasar una lista a una cola

$$listaACola :: [a] \to Cola\ a$$

(E).– Define funciones para mezclar colas y para ordenar colas

$$
\begin{array}{l}
mezcla :: Ord\ a \Rightarrow Cola\ a \to Cola\ a \to Cola\ a \\
ordena :: Ord\ a \Rightarrow Cola\ a \to Cola\ a
\end{array}
$$

(F).– Define una función para *separar* una cola en dos subcolas, tomando los elementos en forma alternada con respecto al orden de la cola original.

18.2. AÑO 1997

18.17 (PD: Febrero, 97) Consideremos las estructuras de datos

$$
\begin{array}{l}
\textbf{data}\ Urna\ a \ = \ V \mid U\ a\ (Urna\ a)\ \textbf{deriving}\ Show \\
\textbf{type}\ Saco\ a \ = \ [\ (a, Int)\ ]
\end{array}
$$

para representar colecciones de objetos (en los sacos, las repeticiones se indican con el entero), y la función de plegado de urnas:

$$
\begin{array}{lll}
pl & :: & (a \rightarrow b \rightarrow b) \rightarrow b \rightarrow (Urna\ a) \rightarrow b \\
pl\ f\ z\ V & = & z \\
pl\ f\ z\ (U\ x\ u) & = & f\ x\ (pl\ f\ z\ u)
\end{array}
$$

(A).– Define la función $mapUrna$ que aplica la función argumento a cada objeto de la urna segundo argumento (map, pero para urnas)

$$mapUrna :: (a \rightarrow b) \rightarrow Urna\ a \rightarrow Urna\ b$$

(B).– Define una función $mapUrna'$ que haga los mismo pero utilizando únicamente el plegado anterior.
(C).– Expresa cómo se aplica el principio de inducción para probar que las funciones $mapUrna$ y $mapUrna'$ son iguales.

(D).– Demuestra, utilizando (C), que efectivamente son iguales.

(E).– Define (utilizando plegados de urnas y de listas) funciones de transformación entre urnas y sacos (que conserven la información) y el cálculo del número de objetos

$$
\begin{array}{lll}
sacoAUrna & :: & Saco\ a \rightarrow Urna\ a \\
cardinalSaco & :: & Saco\ a \rightarrow Int \\
urnaASaco & :: & Eq\ a \Rightarrow Urna\ a \rightarrow Saco\ a \\
cardinalUrna & :: & Urna\ a \rightarrow Int
\end{array}
$$

(F).– Estudia qué calcula la función $cosa$ determinada por la ecuación

$$cosa\ y\ =\ pl\ (\lambda\ x\ u\ \rightarrow\ x == y\ ||\ u)\ False$$

(G).– Estudia qué calcula la función $asco$ determinada por la ecuación

$$asco\ u\ =\ pl\ (\lambda\ x\ t\ \rightarrow\ x\ `cosa`\ u\ \&\&\ t)\ True$$

(H).– Estudia qué calcula la función $extrae$ determinada por la definición

$$
\begin{array}{l}
extrae\ (U\ x\ V) = (x, V) \\
extrae\ (U\ x\ u) \\
\quad |\ x == y \quad\quad = (x, u') \\
\quad |\ otherwise \quad = (y,\ U\ x\ u') \\
\quad \textbf{where} \\
\quad\quad (y, u')\ =\ extrae\ u
\end{array}
$$

18.18 (PD: Febrero, 97) Dibuja el gráfico de una red de procesos y su expresión funcional correspondiente para generar la sucesión:

$$[1,\ 2,\ 4,\ 7,\ 11,\ 16,\ 22,\ 29,\ 37,\ \ldots$$

18.19 (PD: Febrero, 97) **(A).–** Da ejemplos de funciones con los tipos que se indican

$$eje1 :: a \to (b \to a)$$
$$eje2 :: a \to (b \to b)$$

(B).– Infiere el tipo de las funciones de ecuaciones

$$f_1\ x\ y = y\ x\ x$$
$$f_2 = f_1 . f_1$$

18.20 (PD: Febrero, 97, y Setiembre, 97) **(A).–** Describe funciones para implementar predicados con cuantificadores universal ($\forall$) y existencial ($\exists$):

$$paraTodo :: [a] \to (a \to Bool) \to Bool$$
$-- paraTodo\ xs\ p \equiv$ todo elemento de la lista xs verifica p

$$existeAlgún :: [a] \to (a \to Bool) \to Bool$$
$-- existeAlgún\ xs\ p \equiv$ existe al menos un elemento de la lista xs
$--$ verificando p

(B).– Escribe funciones para comprobar las siguientes propiedades que puede verificar un conjunto G (representado mediante una lista) junto con una operación binaria op

$$asociativa :: Eq\ a \Rightarrow [a] \to (a \to a \to a) \to Bool$$
$-- asociativa\ g\ op \equiv$ la operación op es asociativa sobre g

$$conmutativa :: Eq\ a \Rightarrow [a] \to (a \to a \to a) \to Bool$$
$-- conmutativa\ g\ op \equiv$ la operación op es conmutativa sobre g

$$neutroIzda :: Eq\ a \Rightarrow a \to [a] \to Bool$$
$-- neutroIzda\ e\ xs\ g \equiv e$ es elemento neutro a la izquierda ...

$$neutroDcha :: Eq\ a \Rightarrow a \to [a] \to Bool$$
$-- neutroDcha\ e\ xs\ g \equiv e$ es elemento neutro a la derecha ...

$$divisoresDe :: Eq\ a \Rightarrow a \to [a] \to (a \to a \to a) \to [a]$$
$-- divisoresDe\ x\ xs\ op \equiv$ los elementos verifican $y\ `op`\ y' == x$

$$neutro :: Eq\ a \Rightarrow a \to [a] \to (a \to a \to a) \to Bool$$
$-- neutro\ e\ xs\ g \equiv e$ es elemento neutro ...

$$inverso :: Eq\ a \Rightarrow a \to a \to [a] \to (a \to a \to a) \to Bool$$
$$--\ inverso\ x\ x'\ gs\ op \equiv x \text{ es el inverso de } x' \ldots$$

$$grupo :: Eq\ a \Rightarrow (a \to a \to a) \to [a] \to Bool$$
$$--\ grupo\ op\ g \equiv (g, op) \text{ tiene estructura de grupo}$$

(C).– Escribe la función–operador correspondiente a la siguiente tabla, y comprueba las funciones del apartado anterior

$<+>$	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	0	2
3	0	3	2	1

18.21 (LabIV: Junio, 97) **(A)**.– Define una operación $separar :: [a] \to ([a], [a])$ que separe los elementos de una lista en dos listas, una con los elementos que ocupan las posiciones impares y la otra con los que ocupan las posiciones pares.

Define también la operación inversa $reunir :: ([a], [a]) \to [a]$ que reconstruya una lista a partir de otras dos tomando la primera como la lista de las posiciones impares y la segunda como la lista de las posiciones pares del resultado:

$$separar[2,9,3,4,5] \implies ([2,3,5],[9,4])$$
$$reunir\ ([4,3,5],[2,1,0]) \implies [4,2,3,1,5,0]$$

(B).– Utiliza las funciones anteriores para definir una función $maxmin$ que reemplace todos los elementos de las posiciones impares de una lista por el mayor de ellos y todos los de las posiciones pares por el menor de ellos:

$$maxmin\ [2,9,3,4,5] \implies [5,4,5,4,5]$$

(C).– Considerando el tipo árbol siguiente

$$\mathbf{data}\ \acute{A}rbolH\ a = H\ a \mid N\ (\acute{A}rbolH\ a)\ (\acute{A}rbolH\ a)\ \mathbf{deriving}\ Show$$

definir una función $mkárbol :: [a] \to \acute{A}rbolH\ a$ que construya un árbol equilibrado en peso a partir de una lista, utilizando la operación $separar$. (Un árbol está equilibrado en peso cuando la diferencia entre el número de hojas a la izquierda y a la derecha de cada nodo no supera la unidad).

(D).– Define una función $elemento :: \acute{A}rbolH\ a \to Int \to a$ que aplicada a un árbol construido siguiendo el procedimiento del apartado anterior y a un número n, busque en el árbol el elemento que ocupaba la posición n de la lista (comenzando desde 0):

$$elemento\ (mkárbol\ [3,4,5,7,2])\ 2 \implies 5$$

18.22 (LabIV: Junio, 97) Da los tipos y di qué cómputo producen las siguientes funciones :

$$\begin{array}{l} añade1 \quad :: \_\_\_\_ \\ añade1\ x\ y \ = \ concat\ [\ x +\!\!+ y \mid \_ : \_ \leftarrow x\ ] \end{array}$$

$$\begin{array}{l} añade2 \quad :: \_\_\_\_ \\ añade2\ x\ y \ = \ concat\ [\ x\ :\ y \mid \_ : \_ \leftarrow [x]\ ] \end{array}$$

$$\begin{array}{l} añade3 \quad :: \_\_\_\_ \\ añade3\ x\ y \ = \ concat\ [\ x +\!\!+ y \mid \_ : \_ \leftarrow [x]\ ] \end{array}$$

18.23 (LabIV: Junio,97) **(A)**.– Escribe una función para calcular la unión de dos conjuntos interpretados como una lista de elementos no repetidos.
(B).– Dado el tipo $ExpBool$ para representar proposiciones booleanas con las conectivas Y (conjunción), O (disyunción), NG (negación) y AF (afirmación)

$$\begin{array}{l} \mathbf{data}\ ExpBool \ = \ Y\ ExpBool\ ExpBool \\ \qquad\qquad\qquad\quad \mid O\ ExpBool\ ExpBool \\ \qquad\qquad\qquad\quad \mid AF\ Char \\ \qquad\qquad\qquad\quad \mid NG\ Char\ \mathbf{deriving}\ Show \end{array}$$

podemos representar expresiones booleanas como estructuras de este tipo. Así,

$$ejemplo \ = \ Y\ (O\ (AF\ 'p')\ (AF\ 'q'))\ (NG\ 'p')$$

representa la fórmula $(p \vee q) \wedge (\neg p)$. Escribe un predicado $variables$ que devuelva la lista de todos los nombres de proposiciones (sin repetir) que aparecen en una expresión de tipo $ExpBool$:

$$variables\ ejemplo \implies ['p',' q']$$

(C).– Para el tipo anterior, define un predicado $evalúa$ que tome una expresión de tipo $ExpBool$ y una lista de variables y devuelva la evaluación de la expresión sabiendo que las variables que aparecen en la lista deben ser interpretadas como proposiciones verdaderas. Por ejemplo:

$$\begin{array}{ll} evalúa\ ejemplo\ ['p',' q'] & \implies False \\ evalúa\ ejemplo\ ['q'] & \implies True \end{array}$$

(D).– Define la función $partes$ que calcula las partes de un conjunto. En términos de ésta, define la función $tautología$ que indique si una expresión es siempre verdadera para cualquier valor de sus variables proposicionales

18.24 (LabIV: Junio,97) Consideremos el esquema de reducción

$$\begin{array}{l} red\ f\ h\ z\ [\,] \quad\ = h\ z \\ red\ f\ h\ z\ (x : xs) = f\ (h\ x)\ (red\ f\ h\ z\ xs) \end{array}$$

(A).– Infiere el tipo de red.
(B).– Demuestra la siguiente propiedad

$$\forall\ xs\ \centerdot\ xs\ ::\ a\ \centerdot\ 1\ +\ lon\ xs\ =\ red\ (+)\ (\lambda\ x \rightarrow\ 1)\ 0\ xs$$

donde

$$\begin{array}{l} lon\ [\,] \quad\ = 0 \\ lon\ (x : xs) = 1 + lon\ xs \end{array}$$

(C).– Expresa red a partir de $foldr$.
(D).– Define, utilizando red, funciones para: (1) calcular la disyunción de una lista de valores booleanos, y (2) calcular la suma de los elementos de una lista.

18.25 (LabIV: Junio,97) Construye una red de procesos que genere la sucesión de pares

$$(1,0),\ (0,-2),\ (-3,0),\ (0,4),\ (5,0),\ (0,-6),\ (-7,0),\ \ldots$$

Observa que son los puntos de corte con los ejes de coordenadas de una *pseudo–espiral*.

18.26 (PDI y PD: Setiembre, 97) **(A)**.– Infiere el tipo de la función de ecuación $s\ x\ y\ z\ =\ (x\ z)\ (y\ z)$ y da un ejemplo de una función cuyo tipo sea

$$(a\ \rightarrow\ b) \rightarrow\ (a\ \rightarrow\ c)\ \rightarrow\ a\ \rightarrow\ (b, c)$$

18.27 (PDI y PD: Setiembre, 97) Dado el tipo de datos

$$\mathbf{data}\ ExpB\ =\ F\ ExpB\ ExpB \mid N\ ExpB \mid P\ Char\ \mathbf{deriving}\ Show$$

(A).– Construye una función $numVar$ que cuente el número de caracteres que hay en un dato de este tipo:

$$numVar\ ::\ ExpB\ \rightarrow\ Int$$

(B).– Dada la función de plegado $plexpb$ para datos de tipo $ExpB$ definida por:

$$\begin{array}{l} plexpb\ f\ g\ h\ (P\ x) \quad\ = h\ x \\ plexpb\ f\ g\ h\ (N\ e) \quad\ = g\ (plexpb\ f\ g\ h\ e) \\ plexpb\ f\ g\ h\ (F\ e1\ e2) = f\ (plexpb\ f\ g\ h\ e1)\ (plexpb\ f\ g\ h\ e2) \end{array}$$

escribe una función $numVar'$ igual a $numVar$ pero utilizando este plegado.
(C).– Expresa cómo se aplica el principio de inducción para probar la siguiente propiedad:

$$\forall\ ex\ \centerdot\ ex\ ::ExpB\ \centerdot\ numVar\ ex\ =\ numVar'\ ex$$

(D).– Demuestra la propiedad anterior utilizando inducción estructural.

18.28 (PDI y PD: Setiembre, 97) **(A)**.– Construye una red de procesos y su expresión correspondiente para la sucesión infinita :

$$\begin{array}{l}[\ \ (1,\ \ 2,\ \ \ \ 3),\\ \ \ (2,\ \ 2*1,\ \ 1+2),\\ \ \ (3,\ \ 2*2,\ \ 2+2),\\ \ \ (4,\ \ 2*3,\ \ 3+4),\\ \ \ (5,\ \ 2*4,\ \ 4+6),\ldots]\end{array}$$

(B).– Construye una expresión para la lista de los elementos de las tuplas anteriores :

$$[1,\ 2,\ 3,\ 2,\ 2,\ 3,\ 3,\ 4,\ 4,\ 4,\ 6,\ 7,\ 5,\ 8,\ 10,\ 6,\ 10,\ 13,\ \ldots]$$

18.29 (LabV: Setiembre, 97) **(A)**.– Se consideran las matrices de objetos de tipo base a representadas como listas de listas $[[a]]$; completa la siguiente función que calcula la lista de las diagonales (secundarias) de una matriz devolviendo la lista de éstas:

$$\begin{array}{lcl} diagonalesS & :: & [[a]] \rightarrow [[a]] \\ diagonalesS\ [f] & = & \_\_\_\_ \\ diagonalesS\ (f : fs) & = & juntar\ (tail\ f)\ (diagonalesS\ fs) \end{array}$$

$$\begin{array}{lcl} juntar\ [\,]\ \ \ \ ys & = & ys \\ juntar\ (u : us)\ (y : ys) & = & \_\_\_\_ \end{array}$$

Por ejemplo:

$$\begin{array}{l} diagonalesS\ [\ [1,2,3],\\ \qquad\qquad\quad [4,5,6],\\ \qquad\qquad\quad [7,8,9]\ ] \\ \Longrightarrow \\ [[1],[4,2],[7,5,3],[8,6],[9]\ ] \end{array}$$

¿ Cuál es el tipo de la función $juntar$?

(B).– Completa la siguiente función que calcula la imagen especular de una matriz

$$\begin{array}{l} espejo\ ::\ [[a]] \rightarrow [[a]] \\ espejo\ =\ map\ \_\_\_\_ \end{array}$$

Por ejemplo:

$$\begin{array}{l} espejo\ [\ [1,2,3],\\ \qquad\quad [4,5,6],\\ \qquad\quad [7,8,9]\ ] \\ \Longrightarrow \\ [\ [3,2,1],[6,5,4],\ [9,8,7] \end{array}$$

Utilizando la función anterior, define una función para encontrar las diagonales principales

$$\begin{array}{l} diagonalesP :: [[a]] \rightarrow [[a]] \\ diagonalesP\ xs = \_\_\_\_ \end{array}$$

Por ejemplo:

$$\begin{array}{l} diagonalesP\ [\ [1,2,3], \\ \qquad\qquad\quad [4,5,6], \\ \qquad\qquad\quad [7,8,9]] \\ \Longrightarrow \\ [[3],[6,2],[9,5,1],[8,4],[7]] \end{array}$$

(C).– Describe una función que tome una sopa de letras (de tipo $[[Char]]$) y un diccionario de palabras (de tipo $[String]$) y calcule la sopa de palabras (de tipo $[String]$) que consiste en todas las palabras del diccionario que aparecen en la sopa de letras, donde cada palabra puede aparecer en cualquier diagonal, fila o columna así como en cualquier sentido

$$\begin{array}{l} \textbf{type}\ SopaDeLetras = [[Char]] \\ \textbf{type}\ Diccionario = [String] \\ \textbf{type}\ PalabrasEnSopa = [String] \end{array}$$

$$\begin{array}{l} extraePalabras :: SopaDeLetras \rightarrow \\ \qquad\qquad\qquad\quad Diccionario \rightarrow \\ \qquad\qquad\qquad\quad PalabrasEnSopa \end{array}$$

18.30 (LabIV: Setiembre, 97) Dadas las siguientes definiciones para representar vectores y matrices

$$\begin{array}{l} \textbf{type}\ Escalar = Float \\ \textbf{data}\ Vector = MkVec\ [Escalar]\ \textbf{deriving}\ Show \end{array}$$

$$\begin{array}{l} \textbf{type}\ Fila = [Escalar] \\ \textbf{data}\ Matriz = MkMat\ [Fila]\ \textbf{deriving}\ Show \end{array}$$

$$\begin{array}{l} m1 :: Matriz \\ m1 = MkMat\ [[1.0,\ 2.0,\ 3.0\ ],[4.0,\ 5.0,\ 6.0\ ]] \end{array}$$

$$\begin{array}{l} v1 :: Vector \\ v1 = MkVec\ [10.0,\ 20.0,\ 30.0] \end{array}$$

(A).– Escribe una función que devuelva $True$ si la matriz que toma como argumento es válida (i.e., si las longitudes de todas sus filas son iguales):

$$esMat :: Matriz \rightarrow Bool$$

Escribe funciones

$$negarMat :: Matriz \to Matriz$$
$$porMat :: Escalar \to Matriz \to Matriz$$

que cambien el signo y que multipliquen por cierto escalar todos los elementos en una matriz.
(B).– Escribe la función

$$mapMat :: (Escalar \to Escalar) \to Matriz \to Matriz$$

que aplica la función primer argumento a todos los elementos de la matriz segundo argumento. Escribe las funciones $negarMat'$ y $porMat'$ utilizando $mapMat$.
(C).– Escribe una función (detectando posibles errores) que multiplique una matriz por un vector columna:

$$porMixto :: Matriz \to Vector \to Vector$$

Escribe una función (detectando posibles errores) que sume dos matrices:

$$sumMat :: Matriz \to Matriz \to Matriz$$

(D).– Escribe una función de plegado de matrices con el siguiente tipo:

$$foldMat :: (Fila \to a \to a) \to a \to Matriz \to a$$

Escribe una función que devuelva la fila de mayor módulo utilizando la función de plegado anterior:

$$mayorMod :: Matriz \to Fila$$

18.3. Año 1998

18.31 (PDI: Febrero, 98) **(A)**.– Consideremos la siguiente función de *reconstrucción de listas*:

$$\begin{array}{l} recons\ f\ g\ [x] \quad\quad = [g\ x] \\ recons\ f\ g\ (x : xs) = f\ x\ q\ :\ qs \\ \quad \textbf{where} \\ \quad\quad qs@(q : \_) = recons\ f\ g\ xs \end{array}$$

(A).– Deduce su tipo.
(B).– Expresa cómo se aplica el principio de inducción para probar la siguiente propiedad:

$$\forall\, f, g, xs \ .\ xs :: [a],\ xs \neq [\,] \ .\ length\ (recons\ f\ g\ xs) = length\ xs$$

(C).– Demuestra la propiedad anterior utilizando inducción estructural sobre listas.
(D).– Utilizando la función $recons$ define las siguientes funciones para listas no vacías:

```
suma :: Num a ⇒ [a] → a
  -- la suma de los elementos de la lista argumento

máximo :: Ord a ⇒ [a] → a
  -- el mayor elemento de una lista

reemPorÚltimo :: [a] → [a]
  -- reemplaza todos los elementos de la lista argumento
  -- por el último de la lista:
  -- reemPorÚltimo [1,6,3,9] ⟹ [9,9,9,9]

numera :: [a] → [(Int, a)]
  -- numera los objetos de una lista en orden decreciente
  -- numera [5.0,−2.1,5.6] ⟹ [(3,5.0),(2,−2.1),(1,5.6)]
```

(E).– Si representamos un polinomio de una variable $p(x) \equiv a_0 + a_1x + a_2x^2 + \cdots + a_nx^n$ con la lista de sus coeficientes en orden creciente $[a_0, a_1, \ldots, a_n]$, utilizando la función *recons* y la regla de Ruffini escribe una función que calcule el valor $p\ x_0$ de un polinomio en un punto x_0, así como los coeficientes de la división por $x - k$

```
type Polinomio a = [a]
ruffini :: Num a ⇒ a → Polinomio a → (a, Polinomio a)
```

(F).– Y finalmente, de nuevo utilizando la función *recons*, escribe una función que calcule el *segundo menor* elemento de un lista

```
segundoMenor :: Ord a ⇒ [a] → a
  -- segundoMenor[1,6,4,2] ⟹ 2
  -- segundoMenor [3.4]    ⟹ 3.4
```

18.32 (LabV: Febrero, 98) **(A).**– Completa las siguientes definiciones de funciones

```
primo :: Int → Bool
  -- primo n == el número n es primo
primo 1 = False
primo n = ____

divisoresPrimos :: Int → [Int]
  -- divisoresPrimos n == la lista de divisores primos de n
divisoresPrimos n = [ p | p ← [2 .. n], ____ ]
```

(B).– Se pretende escribir una función para obtener la descomposición en factores primos de un número n, resultando una lista de pares (p_i, e_i) de forma que $n = p_1^{e_1} \cdot p_2^{e_2} \cdot \cdots \cdot p_k^{e_k}$ donde cada p_i es un número primo y cada e_i es el grado o exponente con que aparece cada primo. Se pide completar las funciones

```
gradoDe :: Int → Int → Int
  -- el grado con que aparece p en la descomposición
  -- en factores primos de n
```

```
gradoDe n p
  | n `mod` p /= 0 = ____
  | ____           = ____
factorizacion :: Int -> [(Int,Int)]
factorizacion n = ____
```

(C).– Utilizando las funciones anteriores, escribe una función mcd que determine el máximo común divisor de dos números enteros. Nota: El máximo común divisor de dos números se puede obtener a partir de la descomposición en factores primos, tomando los factores comunes con menor exponente.

```
mcd :: Int -> Int -> Int
mcd a b = ____
```

18.33 (LabV: Febrero, 98) Demuestra por inducción estructural

$$long\ .\ map\ f\ =\ long$$

donde *long* es la función

```
long []     = 0
long (x:xs) = 1 + long xs
```

18.34 (PD: Febrero, 98) **(A)**.– Describe una red de procesos (así como su función correspondiente) para encontrar la lista infinita correspondiente a la sucesión $\{a_n\}$ solución de la recurrencia

$$a_{n+3} = 3(n+2)a_{n+2} + 2(n+1)a_{n+1} + na_n, \qquad a_0 = a_1 = a_2 = 1$$

(B).– Utiliza la red anterior para determinar el menor valor de n verificando $a_n + a_{n-1} - a_{n-2} > 100000000$.

18.35 (PDI: Junio, 98) **(A)**.– Deduce el tipo de la siguiente función utilizando el algoritmo de inferencia de tipos:

```
k x y z = x (y z) z
```

(B).– Da dos ejemplos distintos de funciones cuyo tipo sea:

$$(b \to a \to c) \to (b \to c) \to (a \to b) \to (a \to c)$$

18.36 (PDI: Junio, 98) **(A)**.– Encuentra para qué función *f* y para qué valor z es cierto que:

$$map\ (+1).([3]{+\!\!+}\ )\ =\ foldl\ f\ z$$

(B).– Expresa cómo se aplica el principio de inducción estructural para demostrar que las siguientes funciones son equivalentes:

$$g \;=\; map\ (+1)\ .\ inv$$
$$g' \;=\; foldr\ (\lambda\ e\ r\ \rightarrow\ r +\!\!+ [e+1])\ [\,]$$

donde

$$inv\ [\,] \;=\; [\,]$$
$$inv\ (x : xs) \;=\; inv\ xs +\!\!+ [x]$$

(C).– Demuestra $g \;=\; g'$ utilizando la siguiente propiedad

$$(*) \qquad \forall\ f :: a \rightarrow\ b\ \centerdot\ \forall\ u, v :: [a]\ \centerdot\ map\ f\ (u +\!\!+ v) \;=\; map\ f\ u +\!\!+ map\ f\ v$$

18.37 (PDI: Junio, 98) **(A).**– Escribe una función:

$$parDropWhile\ ::\ (a \rightarrow\ a \rightarrow\ Bool)\ \rightarrow\ [a]\ \rightarrow\ [a]$$

de manera que vaya eliminando la cabeza de la lista segundo argumento, mientras los dos primeros elementos de dicha lista verifiquen el predicado primer argumento:

MAIN> $parDropWhile\ (\lambda\ x\ y\ \rightarrow\ x + (y * 2)\ <\ 10)\ [1..]$
$[3..]$

MAIN> $parDropWhile\ (\lambda\ x\ y\ \rightarrow\ x\ <\ y)\ [3,4,5,3,9]$
$[5,3,9]$

MAIN> $parDropWhile\ (\lambda\ x\ y\ \rightarrow\ x\ <\ y)\ [3,4,5,8,9]$
$[\,]$

(B).– Escribe la función $parDropUntil$ (con el mismo tipo que la función $parDropWhile$) que elimina la cabeza de la lista segundo argumento mientras los dos primeros elementos de dicha lista no verifiquen el predicado primer argumento.

18.38 (PDI: Junio, 98) Escribe una red de procesos y la expresión HASKELL correspondiente a la lista:

$$[[1], [0,2], [-1,1,1,3], [-2,0,0,2,0,2,2,4], ...]$$

donde una lista resulta de la anterior sustituyendo cada elemento por su antecesor y su sucesor, es decir, el 1 por el 0 y el 2, el 0 por el –1 y el 1, el 2 por el 1 y el 3, etc.

18.39 (LabIV: Junio, 98) Sean las siguientes declaraciones de tipos para representar polinomios de variable real:

$$\textbf{type}\ Grado \;=\; Int$$
$$\textbf{type}\ Coeficiente \;=\; Float$$
$$\textbf{type}\ Monomio \;=\; (Coeficiente,\ Grado)$$
$$\textbf{infixr}\ 9\ :+:$$
$$\textbf{data}\ Polinomio \;=\; PoliNulo \mid Monomio\ :+:\ Polinomio\ \textbf{deriving}\ Show$$

Supondremos que los monomios de un polinomio están ordenados decrecientemente según sus potencias y simplificados (no aparecen monomios con coeficiente cero). Por ejemplo:

$$p1,\ p2\ ::\ Polinomio$$
$$p1\ =\ (5.0,\ 2)\ :+:\ (7.0,\ 1)\ :+:\ (2.0,\ 0)\ :+:\ PoliNulo$$
$$p2\ =\ (8.0,\ 3)\ :+:\ (9.0,\ 1)\ :+:\ PoliNulo$$

(A).– Define una función $mapPoli$ (*map* actuando sobre $Polinomio$) con el siguiente tipo:

$$mapPoli\ ::\ (Monomio\ \rightarrow\ Monomio)\ \rightarrow\ Polinomio\ \rightarrow\ Polinomio$$

$$\text{MAIN}>\ mapPoli\ (\lambda\ (c, g) \rightarrow (c, g + 1))\ p1$$
$$(5.0, 3)\ :+:\ (7.0, 2)\ :+:\ (2.0, 1)\ :+:\ PoliNulo$$

(B).– Define una función $evalPoli$ que evalúe un polinomio para un valor de la variable, con el siguiente tipo

$$evalPoli\ ::\ Float\ \rightarrow\ Polinomio\ \rightarrow\ Float$$

(C).– Considérese la siguiente función de plegado de polinomios

$$pliegaPoli\ ::\ (Monomio\ \rightarrow\ a\ \rightarrow\ a)\ \rightarrow\ a\ \rightarrow\ Polinomio\ \rightarrow\ a$$
$$pliegaPoli\ f\ e\ PoliNulo\ \ \ \ =\ e$$
$$pliegaPoli\ f\ e\ (m\ :+:\ p)\ =\ f\ m\ (pliegaPoli\ f\ e\ p)$$

Escribe, usando la función de plegado, una función $evalPoli'$ que se comporte como la función del apartado (B).

(D).– Define un operador $<+>$ que obtenga el polinomio suma de otros dos:

$$\mathbf{infixl}\ 6\ <+>$$
$$(<+>)\ ::\ Polinomio\ \rightarrow\ Polinomio\ \rightarrow\ Polinomio$$

$$\text{MAIN}>\ p1\ <+>\ p2$$
$$(8.0, 3)\ :+:\ (5.0, 2)\ :+:\ (16.0, 1)\ :+:\ (2.0, 0)\ :+:\ PoliNulo$$

(E).– Define un operador $<*>$ que obtenga el polinomio producto de otros dos:

$$\mathbf{infixl}\ 7\ <*>$$
$$(<*>)\ ::\ Polinomio\ \rightarrow\ Polinomio\ \rightarrow\ Polinomio$$

$$\text{MAIN}>\ p1\ <*>\ p2$$
$$(40.0, 5) : + : (56.0, 4) : + : (61.0, 3) : + : (63.0, 2)$$
$$: + : (18.0, 1) : + : PoliNulo$$

18.40 (PDI: Setiembre, 98) Infiere el tipo de la siguiente función utilizando el algoritmo de inferencia de tipos: $f\ g\ x\ =\ x\ :\ f\ g\ (g\ x)$

18.41 (PDI: Setiembre, 98) **(A)**.– Sea la siguiente función HASKELL:

$$f\ h\ p\ xs\ ys\ =\ [\ h\ x\ y\ |\ x\ \leftarrow\ xs,\ p\ x,\ y\ \leftarrow\ ys\ ]$$

¿Cuál es el resultado de evaluar la expresión $f\ (+)\ even\ [1..4][10..14]$?
(B).– Escribe otra definición equivalente de f que NO utilice la sintaxis de listas por comprensión.

18.42 (PDI: Setiembre, 98) **(A)**.– Expresa cómo se aplica el principio de inducción estructural para demostrar la siguiente equivalencia:

$$bs +\!\!+ foldl\ (+\!\!+)\ as\ xs\ =\ foldl\ (+\!\!+)\ (bs +\!\!+ as)\ xs$$

(B).– Demuestra la propiedad anterior suponiendo cierta la asociatividad del operador $+\!\!+$.

18.43 (PDI: Setiembre, 98) **(A)**.– Dibuja una red de procesos y escribe la expresión HASKELL correspondiente a la lista infinita:

$$[0,\ 1,\ 2,\ 3,\ 6,\ 11,\ 20,\ 37,\ 68,\ 125,\ \ldots]$$

donde los tres primeros términos de la lista son 0, 1 y 2, y los demás términos se obtienen sumando los tres que les preceden.
(B).– Utiliza el apartado anterior para definir una función que devuelva la siguiente lista infinita de listas:

$$[\,[2,\ 1,\ 0],\ [3,\ 2,\ 1],\ [6,\ 3,\ 2],\ [11,\ 6,\ 3],\ [20,\ 11,\ 6],\ \ldots\,]$$

18.44 (LabIV: Setiembre, 98) Un *entorno* asocia valores a distintas variables. Sean las siguientes definiciones para representar entornos de variables cuyo valor es de tipo entero:

$$\begin{array}{l}\mathbf{type}\ Valor\ \ \ \ \ \ \ =\ Int\\ \mathbf{type}\ NombreVar\ =\ String\\ \mathbf{type}\ Entorno\ \ \ \ \ =\ [(NombreVar,\ Valor)]\end{array}$$

Por ejemplo, el siguiente entorno asocia el valor 10 a la variable $cont$, 15 a la variable b y 60 a la variable $fact$:

$$\begin{array}{l}ent1\ ::\ Entorno\\ ent1\ =\ [\,(\text{"cont"},\ 10),\ (\text{"b"},\ 15),\ (\text{"fact"},\ 60)\,]\end{array}$$

(A).– Escribe una función

$$buscar\ ::\ Entorno\ \rightarrow\ NombreVar\ \rightarrow\ Valor$$

que devuelva el valor que corresponde a la variable segundo argumento en el entorno primer argumento:

```
MAIN> buscar ent1 "b"
15

MAIN> buscar ent1 "v"
Program error : v no definida
```

(B).– Escribe una función

```
asignar :: Entorno → NombreVar → Valor → Entorno
```

que devuelva el nuevo entorno que se obtiene al asignar a la variable segundo argumento el valor tercer argumento a partir del entorno primer argumento:

```
MAIN> asignar ent1 "b" 25
[("cont", 10), ("b", 25), ("fact", 60)]

MAIN> asignar ent1 "v" 70
[("cont", 10), ("b", 15), ("fact", 60), ("v", 70)]
```

(C).– Sean las siguientes declaraciones para representar expresiones aritméticas en las que intervienen números y variables numéricas:

```
infixl 7 :*
infix 7 :/
infixl 6 :+, :-
data Expr = Const Valor | Var NombreVar
          | Expr :+ Expr | Expr :- Expr
          | Expr :* Expr | Expr :/ Expr deriving Show
expr1 :: Expr
expr1 = Const 3 :* Var "cont" :+ Var "b"      -- 3*cont + b
```

Escribe una función

```
evalExpr :: Entorno → Expr → Valor
```

que devuelva el valor obtenido al evaluar la expresión segundo argumento usando el entorno primer argumento para obtener los valores de las variables:

```
MAIN> evalExpr ent1 expr1
45
```

```
MAIN> evalExpr [("cont", 10)] expr1
Program error : b no definida
```

(D).– Sean las siguientes declaraciones para representar sentencias y programas de un lenguaje imperativo simple:

```
infix 5 :=
data Sentencia = NombreVar := Expr                  -- asignación
               | Inc NombreVar                      -- v := v + 1
                 deriving Show
type Programa = [Sentencia]
prog1 = [ "v" := Const 10,                          -- v := 10;
          "b" := Const 20,                          -- b := 20;
          "c" := Const 5 :* Var "v" :+ Var "b",     -- c := 5*v + b;
          Inc "c" ]                                 -- c := c+1
```

—Escribe una función

$$runSent :: Entorno \rightarrow Sentencia \rightarrow Entorno$$

que devuelva el entorno final que se obtiene al ejecutar una sentencia a partir de un entorno inicial:

```
MAIN> runSent [("v",3),("b",1)] ("v" := Const 5 :* Var "v")
[("v",15),("b",1)]
```

—Escribe una función

$$runProg :: Programa \rightarrow Entorno$$

que devuelva el entorno final obtenido al ejecutar secuencialmente cada una de las sentencias de un programa, partiendo de un entorno vacío:

```
MAIN> runProg prog1
[("v",10), ("b",20), ("c",71)]
```

(E).– Si añadimos el siguiente constructor al tipo $Sentencia$ para representar bucles definidos

$$\ldots \mid For\ (NombreVar, Expr, Expr)\ [Sentencia]$$

amplía el apartado (D) para contemplar esta nueva sentencia:

```
prog2 = [ "sum" := Const 0,                         -- sum := 0;
          For("i", Const 1, Const 5) [              -- FOR i := 1 TO 5 DO
          "sum" := Var "sum" :+ Var "i"             --   sum := sum + i
          ]                                         -- EndFOR
          ]
```

```
MAIN> runProg prog2
[("sum",15), ("i",6)]
```

18.45 (LabV: Setiembre, 98) **(A).**– Escribe una función

$$\textbf{type}\ Posición\ =\ Int$$
$$busca\ ::\ String\ \rightarrow\ String\ \rightarrow\ [Posición]$$

que localice las apariciones de una secuencia patrón (primer argumento) dentro de otra secuencia (segundo argumento). La función debe devolver la lista de posiciones desde donde aparece una coincidencia:

MAIN> *busca* "aaba" "aaabaaba"
$[2, 5]\ ::\ [Int]$

AYUDA.– Escribe las siguientes funciones:

$$prefijo\ ::\ String\ \rightarrow\ String\ \rightarrow\ Bool$$
$$buscaPrefijosDesde\ ::\ Int\ \rightarrow\ String\ \rightarrow\ String\ \rightarrow\ [Posición]$$

donde la primera comprueba si el primer argumento es un prefijo del segundo. La segunda función localiza, desde una posición determinada, los enteros desde donde aparecen los prefijos, de tal forma que la función $busca$ es elemental:

$$busca\ \_\ \ bs\ =\ [\,]$$
$$busca\ ps\ bs\ =\ buscaPrefijosDesde\ 1\ ps\ bs$$

Ejemplos pueden ser:

MAIN> *prefijo* "cbc" "cccbc"
False :: *Bool*

MAIN> *prefijo* "cbc" "cbcxxx"
True :: *Bool*

(B).– Di qué calcula la siguiente función:

$$sorpresa\ \ ::\ [[a]]\ \rightarrow\ [[a]]$$
$$sorpresa\ [f]\ \ =\ map\ (\lambda\ x\ \rightarrow\ [x])\ f$$
$$sorpresa\ (f\ :\ fs)\ =\ pegar\ f\ ([\,]\ :\ sorpresa\ fs)$$

$$pegar\ [\,]\ \ \ ds\ \ \ =\ ds$$
$$pegar\ (a\ :\ as)\ (d\ :\ ds)\ =\ (d +\!\!+\ [a])\ :\ pegar\ as\ ds$$

18.4. AÑO 1999

18.46 (PD: Febrero, 99)
(A).– Da ejemplos de funciones con los tipos que se indican

$$eje1\ ::\ (a\ \rightarrow\ b)\ \rightarrow\ a\ \rightarrow\ b \qquad eje2\ ::\ a\ \rightarrow\ (a\ \rightarrow\ b)\ \rightarrow\ b$$

(B).– Infiere el tipo de la función dup que verifica la ecuación $dup\ g\ x\ =\ dup\ g\ (g\ x)$

(C).– Comprueba que la función $dup2$ definida en la forma

$$dup2 \;=\; dup \;.\; dup$$

tiene el mismo tipo que la función anterior

18.47 (PD: Febrero, 99) Sean las funciones:

$$\begin{array}{l} cur \;=\; foldr\ inc\ [\,] \\ foldr \;::\; (a \;\to\; b \;\to\; b\,) \;\to\; b \;\to\; [a] \;\to\; b \\ foldr\ f\ z\ [\,] \qquad\quad =\; z \\ foldr\ f\ z\ (x : xs) \;=\; f\ x\ (foldr\ f\ z\ xs) \end{array}$$

$$\begin{array}{l} inc\ x\ [\,] \qquad\quad\;\; =\; [x] \\ inc\ x\ ys'@(y : ys) \;=\; x + y \;:\; ys' \end{array}$$

(A).– Calcula las expresiones

$$cur\ [a] \qquad cur\ [a, b] \qquad cur\ [a, b, c]$$

(B).– Demuestra que para cualquier lista no vacía se tiene

$$head(\ inc\ x\ xs\ ) \;=\; x \;+\; head\ xs$$

(C).– Sea ahora la siguiente función que calcula la suma de los elementos de listas no vacías

$$\begin{array}{l} sum\ [x] \qquad =\; x \\ sum\ (x : xs) \;=\; x + sum\ xs \end{array}$$

Demuestra que se cumple, para listas no vacías,

$$sum\ xs \;=\; head\ (\ cur\ xs)$$

18.48 (PD y LabV: Febrero, 99) Se consideran las siguientes estructuras para describir matrices

$$\begin{array}{l} \mathbf{type}\ Escalar \;=\; Float \\ \mathbf{type}\ Fila \quad\;\; =\; [Escalar] \\ \mathbf{type}\ Matriz \;=\; [Fila] \end{array}$$

(A).– Escribe las siguientes funciones

$unidad \;::\; Int \;\to\; Int \;\to\; Matriz$
– – $unidad\ n\ m \implies$ la matriz $n \times m$ cuyos elementos son todos iguales
– – a 0.0 salvo los de la diagonal principal que son todos iguales a 1.0
$nula \;::\; Int \;\to\; Int \;\to\; Matriz$
– – $nula\ n\ m \implies$ la matriz $n \times m$ cuyos elementos son todos iguales a 0.0
$diagonal \;::\; Matriz \;\to\; Fila$
– – calcula la diagonal principal

(B).– ¿Qué calcula la función siguiente? Justifica la respuesta

$$\begin{aligned} &desconocida :: Matriz \to Matriz \to Matriz \\ &desconocida = zipWith\ (zipWith\ (+)) \end{aligned}$$

(C).– Escribe una función

$$pliegaM :: (Fila \to a \to a) \to (Fila \to a) \to Matriz \to a$$

que realiza un plegado de una matriz no nula, aplicando la función segundo argumento en caso de que la matriz tenga una sola fila.
(D).– Define las siguientes funciones, utilizando cuando sea necesario la función anterior *pliegaM*

$$\begin{aligned} &módulo :: Fila \to Escalar \\ &\quad \text{– – calcula el módulo de un vector} \\ &mayorMódulo :: Matriz \to Fila \\ &\quad \text{– – calcula la fila de mayor módulo} \\ &mapM :: (Escalar \to Escalar) \to Matriz \to Matriz \\ &\quad \text{– – aplica una función a todos los elementos} \end{aligned}$$

18.49 (LabV: Febrero, 99) Se trata de definir la función

$$suma :: [Int] \to [Int] \to [Int]$$

para obtener la representación de la suma a partir de las representaciones en base 10 de dos números; por ejemplo

$$\begin{aligned} &\text{MAIN}> suma\ [1,2,3,4]\ [4,5,6,7] \\ &[5,\ 8,\ 0,\ 1] :: [Int] \end{aligned}$$

$$\begin{aligned} &\text{MAIN}> suma\ [5,6,7,8]\ [5,6,7,8] \\ &[1,\ 1,\ 3,\ 5,\ 6] :: [Int] \end{aligned}$$

(A).– Define una función auxiliar

$$sumaDígitos :: Int \to Int \to Int \to (Int, Int)$$

que devuelva la suma de tres dígitos y el acarreo producido

$$sumaDígitos\ 5\ 7\ 4 \implies (1,6)$$

(B).– Define la función *suma* en el caso en que las dos listas sean de la misma longitud.
(C).– Modifica la función anterior para el caso en que las dos listas sean de longitudes arbitrarias.
AYUDA.– Utiliza la función *replicate* de PRELUDE.

(D).– Aplica las funciones anteriores para resolver el siguiente puzzle: *reemplazar la letras por dígitos en la suma siguiente de forma que sea correcta*

$$\begin{array}{cccc} & A & B & C & D \\ + & D & A & B & C \\ \hline & C & D & A & B \end{array}$$

(E).– Aplica los apartados anteriores para escribir un programa que genere puzzles de forma automática.

18.50 (PDI: Febrero, 99) Infiere el tipo de las siguientes funciones:

$$\begin{array}{l} fix\ f\ x \ = \ f\ x\ (fix\ f)\ x \\ fix2\ f\ x \ = \ 1\ :\ f\ x\ (fix2\ f)\ x \end{array}$$

18.51 (PDI: Febrero, 99) **(A)**.– Expresa cómo se aplica el principio de inducción estructural para demostrar la siguiente equivalencia:

$$z\ `f`\ foldl\ g\ y\ xs\ =\ foldl\ g\ (z\ `f`\ y)\ xs$$

(B).– Demuestra la propiedad anterior suponiendo cierta la siguiente propiedad para f y g:

$$(P1) \qquad x\ `f`\ (y\ `g`\ z)\ =\ (x\ `f`\ y)\ `g`\ z$$

18.52 (PDI: Febrero, 99) **(A)**.– Sea la siguiente declaración de tipo para representar árboles genéricos en HASKELL:

$$\mathbf{data}\, \acute{A}rbolG\ a\ =\ Vacío \mid NodoG\ a\ [\acute{A}rbolG\ a]\ \mathbf{deriving}\ Show$$

Escribe una función de plegado para estos árboles con el siguiente tipo:

$$pliegaÁrbolG\ ::\ b\ \rightarrow\ (a\ \rightarrow\ [b]\ \rightarrow\ b)\ \rightarrow \acute{A}rbolG\ a\ \rightarrow\ b$$

(B).– Completa la declaración de la función $pertenece$ utilizando la función de plegado del apartado anterior; $pertence$ debe permitir comprobar si un dato aparece en un árbol genérico

$$pertenece\ ::\ Eq\ a\ \Rightarrow\ a\ \rightarrow \acute{A}rbolG\ a\ \rightarrow\ Bool$$

18.53 (PDI: Febrero, 99) **(A)**.– La función exponencial puede ser desarrollada mediante la siguiente serie de potencias:

$$e^x = \sum_{i \geq 0} 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots$$

Escribe una función HASKELL que tome como argumento un valor x y devuelva una lista infinita con el valor de cada uno de los términos de la serie evaluados para dicho valor. Utiliza para ello una red de procesos basada en $zipWith$. Dibuja también la red de procesos utilizada.
(B).– Utiliza el apartado anterior para calcular una aproximación al valor del número e. Para ello, se sumarán los términos de la serie hasta que se obtenga un término menor a una diezmilésima.

18.54 (PD: Junio, 99) Sea la siguiente representación de los enteros (O es la letra o mayúscula) y las siguientes operaciones

$$
\begin{array}{l}
\textbf{data}\ Ent\ =\ O \mid S\ Ent \mid P\ Ent\ \textbf{deriving}\ Show \\
\textbf{instance}\ Num\ Ent\ \textbf{where} \\
\begin{array}{lll}
O + & y & = y \\
(S\ x) + y & & = S(x + y) \\
(P\ x) + y & & = P(x + y) \\
x - & (S\ y) & = P(x - y) \\
x - & O & = x \\
x - & (P\ y) & = S(x - y)
\end{array}
\end{array}
$$

(A).– Infiere el tipo de la siguiente función de plegado de enteros :

$$
\begin{array}{lll}
plegar\ f\ g\ z\ O & = & z \\
plegar\ f\ g\ z\ (S\ x) & = & f(plegar\ f\ g\ z\ x) \\
plegar\ f\ g\ z\ (P\ x) & = & g(plegar\ f\ g\ z\ x)
\end{array}
$$

(B).– Demuestra por inducción estructural que se tiene: $(-)\ =\ plegar\ P\ S$.
(C).– Define a partir de la función $plegar$ el operador $+$, así como una función que describa un isomorfismo ($aInt\ ::\ Ent\ \rightarrow\ Int$). Por ejemplo

$$aInt\ (P\ (P\ O)) \Longrightarrow -2 \qquad\qquad aInt\ (S\ (S\ O)) \Longrightarrow 2$$

18.55 (LabV: Junio, 99) Pretendemos simular un juego en el cual se tiran dos dados y se anota la suma de los puntos. Supondremos que los dados son poliedros de $m + 1$ caras cuyas caras están numeradas de 0 a m, donde m es cierta constante definida en el programa (p.e., $m = 21$) ¿Cuál es la suma de puntos más frecuente? Para realizar la simulación necesitaremos algunas funciones auxiliares.
(A).– Utilizaremos como contador de puntos una lista de enteros, en la cual la posición n–ésima memoriza el número de veces que se obtuvo n puntos al tirar dos dados (la posición 0 será la cabeza de la lista). Escribe una función que devuelva un *contador* inicial. Es decir, una lista de $2m + 1$ elementos todos iguales a 0:

$$
\begin{array}{l}
m\ ::\ Int \\
m\ =\ 21 \\
\textbf{type}\ Contador\ =\ [Int] \\
contadorInicial\ ::\ Contador
\end{array}
$$

(B).– El siguiente paso es definir una función que incremente en una unidad cierta posición de un contador. Define una función

$$\begin{array}{l}\textbf{type}\ Posicion\ =\ Int\\ incrementa\ ::\ Posicion\ \rightarrow\ Contador\ \rightarrow\ Contador\\ \quad --\ incrementa\ 3\ [x,y,z,u,t,\ldots]\Longrightarrow\ [x,y,z,u+1,t,\ldots]\end{array}$$

(C).– Dada una función iteradora $f\ ::\ Int\ \rightarrow\ Int$, se pretende generar la lista infinita $[a_0, fa_0, f(fa_0), \ldots]$, donde a partir de un número inicial a_0 se aplica sucesivamente la función iteradora. Si tomamos como función iteradora la función

$$f\ x\ =\ (77 * x + 1)\ `rem`\ 1024$$

entonces se genera una secuencia pseudo-aleatoria de números del intervalo [0..1023]; tal lista se puede transformar en una lista de números de $[0..m]$, para simular la tirada de un dado *infinitas* veces. Escribe una función que devuelva una lista pseudo-aleatoria de tiradas de un dado, a partir de una semilla inicial sem

$$\begin{array}{l}dados\ ::\ Int\ \rightarrow\ [Int]\\ dados\ sem\ =\ \_\_\_\_\end{array}$$

(D).– El siguiente paso es simular el proceso de juego; define una función

$$\begin{array}{l}\textbf{type}\ NúmeroDeTiradas\ =\ Int\\ \textbf{type}\ Semilla\ =\ Int\\ simula\ ::\ NúmeroDeTiradas\ \rightarrow\ Semilla\ \rightarrow\ Contador\end{array}$$

de forma que $simula\ nTirada\ sem$ devuelva un contador que describa el total de veces que salió cada posible puntuación (la suma de las caras de dos dados) al simular un número de tiradas (de dos dados) igual a $nTirada$, teniendo en cuenta que la sucesión de aleatorios se genera desde sem. Por ejemplo, si $m = 3$ (donde $m + 1$ es el número de caras del dado),

$$\begin{array}{l}\text{MAIN}>\ simula\ 9\ 42\\ [0,0,3,5,0,0,1]\end{array}$$

que expresa que en tres ocasiones se obtuvo la suma 2, en cinco ocasiones la suma 3, y en una sola ocasión apareció la suma 6. Ten presente que hay que tirar dos dados.

(E).– Escribe una función para estudiar la diferencia entre la probabilidad y la frecuencia de cada valor; recuerda que la probabilidad de obtener un total de v puntos con dos dados de $m + 1$ caras es

```
frec :: Fractional a => Int -> a
frec v
  | v < m      = fromIntegral(v + 1)/ fromIntegral((m + 1) ↑ 2)
  | otherwise  = fromIntegral(2 * m - v + 1) / fromIntegral((m + 1) ↑ 2)
estudioEstadístico :: NúmeroDeTiradas -> Semilla ->
                      (Contador, [Double])
  -- el primer contador es el resultado del experimento, y
  -- el segundo contador describe los valores teóricos
```

18.56 (PDI: Junio, 99) Considérense las siguientes declaraciones para representar números enteros y operaciones entre éstos en HASKELL:

```
data Ent = O | S Ent | P Ent deriving (Eq, Show)
negar O     = O
negar (S x) = P (negar x)
negar (P x) = S (negar x)
instance Num Ent where
  x + O   = x
  x + S y = S (x + y)
  x + P y = P (x + y)
  x - O   = x
  x - S y = P (x - y)
  x - P y = S (x - y)
```

(A).– Expresa cómo se aplica el principio de inducción estructural para demostrar la siguiente propiedad:

$$\forall\, x,\ y\ ::\ Ent\ \centerdot\ x - y = x + negar\ y$$

(B).– Demuestra la propiedad anterior.

Considérese ahora la siguiente función de plegado de enteros:

```
plegar f g e O     = e
plegar f g e (S x) = f (plegar f g e x)
plegar f g e (P x) = g (plegar f g e x)
```

(C).– Expresa la función $negar$ y los operadores $+$ y $-$ usando $plegar$.
(D).– Define usando $plegar$ un operador que calcule el producto de dos valores de tipo Ent.

18.57 (PDI: Junio, 99) Usa el algoritmo de inferencia de tipos para calcular el tipo de la siguiente función:

```
x y z w = (w z)(z y)
```

18.58 (PDI: Junio, 99) Considérese la siguiente lista infinita:

$$[(0,1),\ (1,1),\ (3,2),\ (6,6),\ (10,24),\ \ldots]$$

que proviene de

$$\begin{array}{l}[\ (0,1),\\ \ (0+1,1*1),\\ \ (0+1+2,1*1*2),\\ \ (0+1+2+3,1*1*2*3),\ \ldots]\end{array}$$

(A).– Dibuja una red de procesos para el cómputo de la lista anterior y escribe el correspondiente programa HASKELL.
(B).– Utiliza el apartado anterior para escribir una función que devuelva la siguiente lista:

$$[(0,1),(1,0),(1,1),(1,1),(3,2),(2,3),(6,6),(6,6),\ \ldots]$$

18.59 (LabIV: Junio, 99) Considérese el siguiente tipo $Dicc$ para representar diccionarios que asocian una cadena de caracteres a un valor:

```
data As a    = A String a deriving Show
type Dicc a  = [As a]
  -- por ejemplo:
ejDicc :: Dicc Int
ejDicc = [A "Mercedes" 3, A "Seat" 2]
```

(A).– Escribe una función $creaDicc\ ::\ Dicc\ a$ que cree un diccionario vacío. Escribe una función

$$actualiza\ ::\ Dicc\ a\ \rightarrow\ String\ \rightarrow\ a\ \rightarrow\ Dicc\ a$$

tal que dado un diccionario, una cadena clave y un valor, actualiza el diccionario colocando para esa clave ese valor. Si no existe la clave, la inserta. El diccionario debe mantener las claves ordenadas. Por ejemplo:

```
MAIN> actualiza ejDicc "Mercedes" 4
[A "Mercedes" 4, A "Seat" 2]

MAIN> actualiza ejDicc "Renault" 3
[A "Mercedes" 3, A "Renault" 3, A "Seat" 2]
```

Escribe la función

$$está\ ::\ Dicc\ a\ \rightarrow\ String\ \rightarrow\ Bool$$

que devuelve $True$ si la clave se encuentra en el diccionario. En otro caso devuelve $False$. Escribe la función

$$valor :: Dicc\ a \rightarrow String \rightarrow a$$

que devuelve el valor asociado a una clave en un diccionario. Si la clave no existe debe producir error. Por ejemplo:

MAIN> *valor ejDicc* "Seat"	MAIN> *valor ejDicc* "Porche"
2	*Program error* : *La clave no existe*

(B).– Dada una lista de palabras, en las que posiblemente aparezcan palabras repetidas, diseña una función

$$cuentaPalabras :: [String] \rightarrow Dicc\ Int$$

que a partir de ella crea un diccionario que mantiene información de las veces que aparece cada palabra en la lista. Por ejemplo, si

$$ejPal = [\text{"hola"}, \text{"que"}, \text{"hola"}, \text{"tal"}, \text{"como"}, \text{"tal"}, \text{"hola"}, \text{"que"}]$$

entonces tendremos el siguiente diálogo

MAIN> *cuentaPalabras ejPal*
$[A$ "como" $1,\ A$ "hola" $3,\ A$ "que" $2,\ A$ "tal" $2]$

(C).– Construye una función

$$porApariciones :: [String] \rightarrow [(String, Int)]$$

que tome listas como la del apartado anterior y produzca como resultado una lista de pares cuya primera componente son las cadenas que se encuentran en la lista y la segunda componente el número de veces que aparece. En este caso, la lista de pares debe estar ordenada según el número de apariciones de mayor a menor. Por ejemplo:

MAIN> *porApariciones ejPal*
$[(\text{"hola"}, 3), (\text{"que"}, 2), (\text{"tal"}, 2), (\text{"como"}, 1)]$

(D).– Se define el tipo *Encuesta* como una lista de pares en la que la primera componente es el número de la pregunta (como cadena de caracteres) y la segunda la respuesta dada en la encuesta a esa pregunta:

$$\textbf{type}\ Encuesta = [(String,\ String)]$$
$$ejen1 = [(\text{"1"}, \text{"nunca"}), (\text{"2"}, \text{"Seat"}), (\text{"3"}, \text{"a veces"}), (\text{"4"}, \text{"no"})]$$
$$ejen2 = [(\text{"1"}, \text{"si"}), (\text{"2"}, \text{"Mercedes"}), (\text{"3"}, \text{"a veces"}), (\text{"5"}, \text{"nunca"})]$$

Una encuesta puede contener cualquier número de preguntas. Por otro lado el que responde a una encuesta puede dejar preguntas sin contestar. Pretendemos crear un diccionario con las siguientes características:

— Cada clave será un número de pregunta de una encuesta

— El valor asociado a una clave será a su vez otro diccionario.

— Este diccionario tendrá como clave las posibles respuestas dadas a esa pregunta, y como valor el número de veces que aparece esa respuesta. Por ejemplo:

$$[A\ \text{"1"}\ [A\ \text{"nunca"}\ 1,\ A\ \text{"si"}\ 3],\ A\ \text{"2"}\ [A\ \text{"Mercedes"}\ 4]]$$

quiere decir que a la pregunta 1 se ha contestado una vez "nunca" y tres veces "si", y a la pregunta 2 se ha contestado cuatro veces "Mercedes". Construye la función

$$acum\ ::\ Encuesta\ \rightarrow\ Dicc\ (Dicc\ Int)\ \rightarrow\ Dicc\ (Dicc\ Int)$$

que toma una encuesta y un diccionario de los mencionados y acumula la encuesta al diccionario devolviendo el diccionario resultante. Por ejemplo:

MAIN> $acum\ ejen2\ [A\ \text{"1"}\ [A\ \text{"nunca"}\ 1],\ A\ \text{"2"}\ [A\ \text{"Mercedes"}\ 1]]$
$[A\ \text{"1"}\ [A\ \text{"nunca"}\ 1,\ A\ \text{"si"}\ 1],\ A\ \text{"2"}\ [A\ \text{"Mercedes"}\ 2],$
$A\ \text{"3"}\ [A\ \text{"a veces"}\ 1],\ A\ \text{"5"}\ [A\ \text{"nunca"}\ 1]]$

Construye una función

$$resultados\ ::\ [Encuestas]\ \rightarrow\ Dicc\ (Dicc\ Int)$$

tal que a partir de una lista de encuestas genere un diccionario como el anterior con toda la información de las encuestas. Por ejemplo:

MAIN> $resultados\ [ejen1,\ ejen2]$
$[A\ \text{"1"}\ [A\ \text{"nunca"}\ 1,\ A\ \text{"si"}\ 1],\ A\ \text{"2"}\ [A\ \text{"Mercedes"}\ 1,\ A\ \text{"Seat"}\ 1],$
$A\ \text{"3"}\ [A\ \text{"a veces"}\ 2],\ A\ \text{"4"}\ [A\ \text{"si"}\ 1],\ A\ \text{"5"}\ [A\ \text{"nunca"}\ 1]]$

18.60 (LabIV: Junio, 99) Considérese el siguiente tipo para representar imágenes monocromas mediante líneas de píxeles, donde un pixel negro se representa con el valor 1 y uno blanco con el 0:

$$\begin{array}{l}
\textbf{type}\ Pixel\ \ \ = Int\\
\textbf{type}\ Línea\ \ \ = [Pixel]\\
\textbf{type}\ Imagen\ = [Línea]\\
triángulo\ ::\ Imagen\\
triángulo\ =\ [\,[1,0,0,0],\\
\qquad\qquad\ [1,1,0,0],\\
\qquad\qquad\ [1,1,1,0],\\
\qquad\qquad\ [1,1,1,1]\,]
\end{array}$$

(A).– Escribe las funciones

$$\begin{array}{l}
intercalar\ ::\ a\ \rightarrow\ [a]\ \rightarrow\ [a]\\
invertir\ \ \ ::\ [a]\ \rightarrow\ [a]
\end{array}$$

La primera inserta un elemento entre cada dos elementos de una lista, mientras que la segunda invierte una lista:

MAIN> $intercalar1[2,3,4]$
$[2,1,3,1,4]$

MAIN> $invertir[2,3,4]$
$[4,3,2]$

(B).– Escribe una función $pinta :: Imagen \rightarrow String$ que devuelva la cadena de caracteres que representa una figura. Los ceros deben ser reemplazados por el carácter punto, los unos por el carácter 'X' y debe aparecer un salto de línea (carácter '\n') entre cada dos líneas de la figura, es decir,

$$pinta\ triángulo \Longrightarrow \text{"X...\nXX..\nXXX.\nXXXX"}$$

MAIN> $putStr(pintatriángulo)$
$X...$
$XX..$
$XXX.$
$XXXX$

MAIN> $(pinta.refH)triángulo$
$...X$
$..XX$
$.XXX$
$XXXX$

Escribe funciones que reflejen horizontalmente ($refH$), verticalmente ($refV$) y en ambas direcciones ($refHV$) una figura.

MAIN> $(pinta.refV)triángulo$
$XXXX$
$XXX.$
$XX..$
$X...$

MAIN> $(pinta.refHV)triángulo$
$XXXX$
$.XXX$
$..XX$
$...X$

(C).– Escribe dos operadores que permitan obtener una figura al unir horizontal y verticalmente dos figuras. Puedes suponer que las figuras a unir tienen el mismo tamaño en la dimensión que se unen. Define dos operadores $\mathbf{infixl}\ 5\ >+<, >-<$ de forma que

MAIN> $pinta\ (triángulo\ >+<\ refHV\ triángulo)$
$X...$
$XX..$
$XXX.$
$XXXX$
$XXXX$
$.XXX$
$..XX$
$...X$

MAIN> $pinta\ (triángulo\ >-<\ refHV\ triángulo)$
$X...XXXX$
$XX...XXX$
$XXX...XX$
$XXXX...X$

(D).– Escribe una función $negar :: Imagen \rightarrow Imagen$, que devuelva una figura *en negativo*:

```
MAIN> (pinta . negar) triángulo
.XXX
..XX
...X
....
```

(E).– Escribe usando la función *foldr* una función

$$repite :: Int \rightarrow [a] \rightarrow [a]$$

que repita un número de veces dado cada elemento de una lista. Por ejemplo,

```
MAIN> repite 3 "abc"
aaabbbccc
```

Escribe una función

$$superponer :: Imagen \rightarrow Imagen \rightarrow Imagen$$

que toma dos figuras (se supone del mismo tamaño) y devuelve otra, también del mismo tamaño pero en la que aparecen las dos figuras superpuestas. Se supone que un 1 en cualquier figura produce un 1 en el resultado, mientras que para producir un 0 es necesario que ambas figuras tengan un 0.

```
MAIN> pinta (superponer triángulo (refH triángulo))
X..X
XXXX
XXXX
XXXX
```

Escribe una función *zoom*, que tome un entero n y una figura, y construya la figura que se obtiene aumentando la original el número de veces indicado. Cada pixel de la figura original dará lugar a $n \times n$ pixeles en la nueva figura.

```
MAIN> pinta(zoom2triángulo)
XX......
XX......
XXXX....
XXXX....
XXXXXX..
XXXXXX..
XXXXXXXX
XXXXXXXX
```

```
MAIN> pinta(zoom3triángulo)
XXX.........
XXX.........
XXX.........
XXXXXX......
XXXXXX......
XXXXXX......
XXXXXXXXX...
XXXXXXXXX...
XXXXXXXXX...
XXXXXXXXXXXX
XXXXXXXXXXXX
XXXXXXXXXXXX
```

18.61 (PD: Setiembre, 99) Sea la siguiente representación de los enteros (O es la letra o mayúscula), las operaciones aritméticas indicadas, y la siguiente función de plegado de enteros :

$$
\begin{array}{l}
\textbf{data}\ Ent\ =\ O \mid S\ Ent \mid P\ Ent\ \textbf{deriving}\ Show \\
\textbf{instance}\ Num\ Ent\ \textbf{where} \\
\quad 0 \quad + y \quad = y \\
\quad (S\ x) + y \quad = S(x + y) \\
\quad (P\ x) + y \quad = P(x + y) \\
\quad x \quad - O \quad = x \\
\quad x \quad - (S\ y) \ = P(x - y) \\
\quad x \quad - (P\ y) \ = S(x - y) \\
plegar\ f\ g\ z\ O \quad = z \\
plegar\ f\ g\ z\ (S\ x) \ = f(plegar\ f\ g\ z\ x) \\
plegar\ f\ g\ z\ (P\ x) \ = g(plegar\ f\ g\ z\ x)
\end{array}
$$

(A).– Estudia los tipos de las siguientes funciones:

$$
\begin{array}{l}
f1\ =\ plegar\ not\ not\ True \\
f2\ =\ plegar\ not\ not\ False \\
f3\ =\ plegar\ S \quad S \quad O
\end{array}
$$

(B).– Di qué calcula cada una de las funciones anteriores, justificando la respuesta.
(C).– Prueba por inducción estructural las siguientes identidades, interpretando cada una de ellas:

$$\forall\ x\ \centerdot\ x :: Ent\ \centerdot \qquad f1\ x\ =\ f1\ (f3\ x) \qquad f1\ =\ not\ f2$$

(D).– Escribe el producto $(*)$ de enteros utilizando la función $plegar$, y demuestra por inducción estructural que se cumple:

$$\forall\ y\ \centerdot\ y\ ::\ Ent\ \centerdot\ (S\ O)\ *\ y\ =\ y$$

18.62 (LabV: Setiembre, 99) Pretendemos describir un $motor$ de inferencia para PROLOG, pero considerando objetivos sin variables. Para ello se consideran las siguientes estructuras de datos para codificar los programas

$$
\begin{array}{l}
\textbf{data}\ Obj \quad = P|Q|R|S|T\ \textbf{deriving}\ (Eq, Show) \\
\textbf{type}\ Secuencia\ = [Obj] \\
\textbf{data}\ Clausula \ = Obj\ :-\ Secuencia\ \textbf{deriving}\ Show \\
\textbf{type}\ Programa\ = [Clausula]
\end{array}
$$

Por ejemplo, el programa que tiene por reglas:

$$
\begin{array}{llll}
p :- q, r. & q. & r. & s. \\
 & q :- s. & & s :- t.
\end{array}
$$

(el objetivo t siempre falla) se codifica con la lista:

$$miPrograma :: Programa$$
$$miPrograma =$$
$$[\, P :- [Q, R],\ Q :- [\,],\ Q :- [S],\ R :- [\,],\ S :- [\,],\ S :- [T]]$$

(A).– Escribe una función

$$éxitos :: Secuencia \rightarrow Int$$

que devuelva el número de éxitos de una secuencia de objetivos. Por ejemplo:

```
MAIN> éxitos [P, T]
0 :: Int

MAIN> éxitos [P]
2 :: Int
```

AYUDA: considérense las siguientes declaraciones incompletas:

```
data Resultado = Éxito | Fallo deriving Show
resuelve       :: Secuencia → [Resultado]
éxitos os      = ____
resuelve []    = [Éxito]
resuelve (o:os) = concat [____ | ____ ]
```

de tal manera que la función *resuelve* devuelve una lista con tantos objetos iguales a *Éxito* como éxitos tenga la llamada (número de ramas con éxito en el árbol de resolución):

```
MAIN> resuelve [P]
[Éxito,Éxito] :: [Resultado]

MAIN> resuelve [P, T]   -- T falla
[] :: [Resultado]

MAIN> resuelve [P, Q]
[Éxito,Éxito,Éxito,Éxito] :: [Resultado]
```

(B).– Ahora modificaremos la función *resuelve* para describir otra función que permita construir las trazas asociadas a la resolución de una secuencia de objetivos. Si consideramos las declaraciones

```
type Traza = [Obj]
type Trazas = [Traza]
```

Una traza será la lista de secuencias de objetivos *pendientes de resolver* hasta cada nodo del árbol de búsqueda. Consideraremos dos funciones:

```
resuelve2 :: Secuencia → Traza → Trazas
trazar    :: Secuencia → Trazas
```

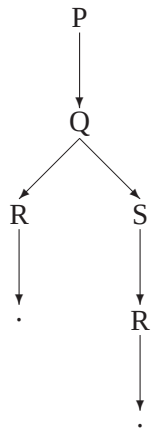

Figura 18.1: Árbol–traza para $resuelve2\ [Q]\ [[T]]$ y para $trazar\ [Q]$..

La primera toma una secuencia de objetivos y una traza inicial y devuelve las posibles trazas a partir de la inicial. La segunda función toma una secuencia inicial y devuelve las trazas posibles con éxito (recorrido con las llamadas a los objetivos que tuvieron éxito). Por ejemplo (ver Figura 18.1):

$$\begin{array}{l}\text{MAIN}> \ resuelve2\ [Q]\ [T,\ T,\ T,\ T] \\ [[T,\ T,\ T,\ T,\ Q],[T,\ T,\ T,\ T,\ Q,\ S]]\ ::\ [[Obj]]\end{array}$$

$$\begin{array}{l}\text{MAIN}> \ resuelve2\ [Q]\ [T] \\ [[T,\ Q],[T,\ Q,\ S]]\ ::\ [[Obj]]\end{array}$$

$$\begin{array}{l}\text{MAIN}> \ trazar\ [Q] \\ [[Q],[Q,\ S]]\ ::\ [[Obj]]\end{array}$$

$$\begin{array}{l}\text{MAIN}> \ trazar\ [P] \\ [[P,\ Q,\ R],[P,\ Q,\ S,\ R]]\ ::\ [[Obj]]\end{array}$$

AYUDA.– Completar el siguiente código:

$$\begin{array}{lll} resuelve2\ [\,] & t & = [t] \\ resuelve2\ (o : os) & t & = concat\ [\ \_\_\_\_\ |\ \_\_\_\_\ ] \\ trazar\ os & & = resuelve2\ \_\_\_\_ \end{array}$$

18.63 (LabV: Setiembre, 99) Para representar circuitos combinacionales $\mathcal{CC}$ en HASKELL (ver Figura 18.2) utilizaremos el siguiente tipo de datos

$$\begin{array}{lll} \textbf{type}\ Retraso & = & Int \\ \textbf{type}\ Bit & = & Int \\ \textbf{type}\ Var & = & String \end{array}$$

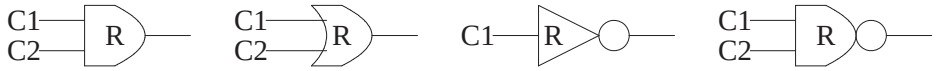

Figura 18.2: Puertas lógicas elementales..

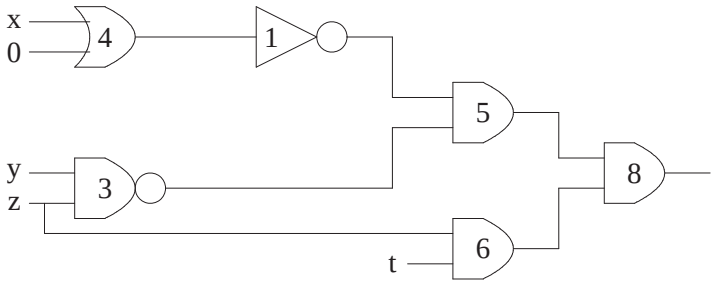

Figura 18.3: Un circuito combinacional..

$$
\begin{array}{lll}
\textbf{data}\ \mathit{Circuito} & = & \mathit{AND}\ \ \mathit{Circuito}\ \mathit{Circuito}\ \mathit{Retraso} \\
& | & \mathit{OR}\ \ \ \mathit{Circuito}\ \mathit{Circuito}\ \mathit{Retraso} \\
& | & \mathit{NAND}\ \mathit{Circuito}\ \mathit{Circuito}\ \mathit{Retraso} \\
& | & \mathit{NOT}\ \ \mathit{Circuito}\ \mathit{Retraso} \\
& | & \mathit{BIT}\ \ \ \mathit{Bit} \\
& | & \mathit{VAL}\ \ \ \mathit{Var}\ \textbf{deriving}\ \mathit{Show}
\end{array}
$$

donde el dato $BIT\ Bit$ es utilizado para representar las entradas con valores fijos (0 o 1). Por otro lado, el dato $VAL\ Var$ es utilizado para representar las entradas con valor variable. En las expresiones $AND\ C_1\ C_2\ R$, ..., los C_i son circuitos y R es un entero positivo que representa el retraso de la puerta. Por ejemplo, el circuito de la Figura 18.3 puede escribirse como el resultado de la función siguiente:

$$
\begin{array}{l}
\mathit{ejemplo}\ = \mathit{AND}\ (\mathit{AND}\ (\mathit{NOT}\ (\mathit{OR}\ (\mathit{VAL}\ \text{"x"})\ (\mathit{BIT}\ 0)\ 4)\ 1) \\
\qquad\qquad (\mathit{NAND}\ (\mathit{VAL}\ \text{"y"})\ (\mathit{VAL}\ \text{"z"})\ 3)\ 5\ ) \\
\qquad\qquad (\mathit{AND}\ (\mathit{VAL}\ \text{"z"})\ (\mathit{VAL}\ \text{"t"})\ 6)\ 8
\end{array}
$$

(A).– Escribe las siguientes funciones HASKELL

$$
\begin{array}{l}
\mathit{maxRetraso}\ ::\ \mathit{Circuito}\ \rightarrow\ \mathit{Retraso} \\
\mathit{entradas}\ ::\ \mathit{Circuito}\ \rightarrow\ [\mathit{Var}]
\end{array}
$$

donde la primera calcula el máximo retraso del circuito que se define como el máximo valor de los retrasos acumulados en cada rama. La segunda función calcula la lista con

los nombres que representan las entradas variables del circuito (sin repeticiones). Por ejemplo

```
MAIN> maxRetraso ejemplo
18

MAIN> entradas ejemplo
["x", "y", "z", "t"]
```

AYUDA.– Utiliza una función para el cálculo de la unión de dos conjuntos.

(B).– Define la función

$$puertas :: Circuito \rightarrow [(String, Int)]$$

que devuelve una lista de pares donde el primer elemento del par indica la puerta utilizada y el segundo el número de veces que se utiliza tal puerta en el $\mathcal{CC}$. Por ejemplo:

```
MAIN> puertas ejemplo
[("not", 1), ("nand", 1), ("or", 1), ("and", 3)]
```

Además, escribe la función *entradas* anterior utilizando la siguiente función de plegado de circuitos

$$\begin{aligned}
foldC :: &(a \rightarrow a \rightarrow Retraso \rightarrow a) \rightarrow \\
&(a \rightarrow a \rightarrow Retraso \rightarrow a) \rightarrow \\
&(a \rightarrow a \rightarrow Retraso \rightarrow a) \rightarrow \\
&(a \rightarrow Retraso \rightarrow a) \rightarrow \\
&(Bit \rightarrow a) \rightarrow \\
&(String \rightarrow a) \rightarrow \\
&Circuito \rightarrow \\
&a
\end{aligned}$$

$$\begin{aligned}
&foldC\ f\ g\ h\ i\ j\ k\ (AND\ c1\ c2\ r) = \\
&\qquad f\ (foldC\ f\ g\ h\ i\ j\ k\ c1)\ (foldC\ f\ g\ h\ i\ j\ k\ c2)\ r \\
&foldC\ f\ g\ h\ i\ j\ k\ (OR\ c1\ c2\ r) = \\
&\qquad g\ (foldC\ f\ g\ h\ i\ j\ k\ c1)\ (foldC\ f\ g\ h\ i\ j\ k\ c2)\ r \\
&foldC\ f\ g\ h\ i\ j\ k\ (NAND\ c1\ c2\ r) = \\
&\qquad h\ (foldC\ f\ g\ h\ i\ j\ k\ c1)\ (foldC\ f\ g\ h\ i\ j\ k\ c2)\ r \\
&foldC\ f\ g\ h\ i\ j\ k\ (NOT\ c1\ r) = i\ (foldC\ f\ g\ h\ i\ j\ k\ c1)\ r \\
&foldC\ f\ g\ h\ i\ j\ k\ (BIT\ b) \quad = j\ b \\
&foldC\ f\ g\ h\ i\ j\ k\ (VAL\ v) \quad = k\ v
\end{aligned}$$

(C).– Describe las siguientes funciones

$$\begin{aligned}
eval &:: Circuito \rightarrow [Var] \rightarrow Bit \\
tablaVerdad &:: Circuito \rightarrow [([Var], Bit)]
\end{aligned}$$

de forma que la expresión *eval c ents* calcule el valor de la salida del circuito c para la entrada *ents*. Este *ents* es una lista con una selección de nombres de variables del

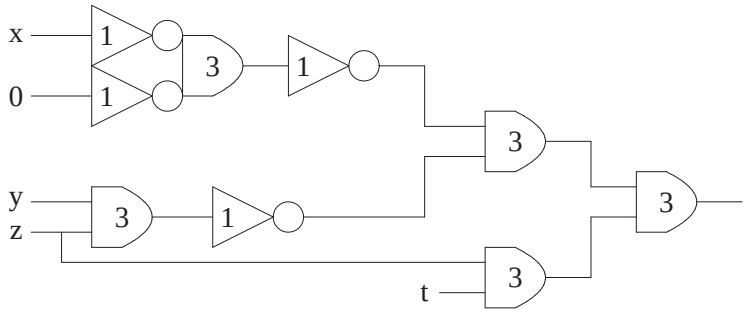

Figura 18.4: Un circuito que usa únicamente puertas and y not..

circuito c. Las variables que aparecen en esta lista tomarán el valor 1 y las que no aparecen el valor 0. Además, la expresión $tablaVerdad\ c$ devuelve una lista con todos las combinaciones de entradas posibles al circuito y su correspondiente valor de salida. Por ejemplo:

```
MAIN> eval ejemplo ["x", "z"]
0

MAIN> eval ejemplo ["z", "t"]
1

MAIN> tablaVerdad ejemplo
[ (["x", "y", "z", "t"], 0), (["z", "t"], 1), ...
```

(D).– Escribe las siguientes funciones:

$$andYNot :: Circuito \rightarrow Circuito$$
$$soloNand :: Circuito \rightarrow Circuito$$

de forma que la expresión $andYNot\ c$ calcula un circuito que sólo emplea puertas *and*(con retraso 3) y *not* (con retraso 1) obtenido a partir de (equivalente a) c (ver Figura 18.4). En forma similar, $soloNand\ c$ calcula un circuito equivalente a c pero construido únicamente con puertas $nand$ (con retraso 5). Se recuerda que una puerta and se consigue con tres $nand$ y una puerta not se consigue con una $nand$. Por ejemplo:

```
MAIN> ejemplo
And (And (Not (And (Not (Val "x") 1) (Not (Bit 0) 1 ) 3 ) ) 1)

MAIN> andYNot ejemplo
(And (Val "x") (Val "z") 3) 3) (Not (And (Val "z") (Val "t") 3) 1) 3)
```

18.5. AÑO 2000

18.64 (PD: Febrero, 2000) (fichero: PD Febrero 00.hs) Sea la siguiente estructura de datos para representar colas *arbóreas*

$$\begin{array}{l}\textbf{infixr}\ 5\ :>\\ \textbf{data}\ Cola\ a\ =\ U\ a\mid Cola\ a\ :>\ Cola\ a\ \textbf{deriving}\ Show\end{array}$$

donde el constructor infijo $(:>)$ es asociativo a la derecha, y por tanto, la interpretación de la cola $U\ 1\ :>\ U\ 2\ :>\ U\ 3$ es $U\ 1\ :>\ (U\ 2\ :>\ U\ 3)$ (el primer elemento de la cola es 1 y el último es 3).

(A).– Define la función que elimina el último elemento de una cola:

$$deCola\ ::\ Cola\ a\ \rightarrow\ (Cola\ a,\ a)$$

de forma que por ejemplo tengamos

$$\begin{array}{l}\text{MAIN}>\ deCola\ (U\ 1\ :>\ U\ 2\ :>\ U\ 3)\\ (U\ 1\ :>\ U\ 2,\ 3)\ ::\ (Cola\ Integer, Integer)\end{array}$$

(B).– Deduce el tipo de la siguiente función de *plegado* de colas

$$\begin{array}{l}redCola\ f\ g\ (U\ x)\ \ \ \ \ \ =\ g\ x\\ redCola\ f\ g\ (c\ :>\ c')\ =\ f\ (redCola\ f\ g\ c)\ (redCola\ f\ g\ c')\end{array}$$

(C).– Estudia qué computa la siguiente función, justificando la respuesta

$$\begin{array}{l}cosa\ ::\ Cola\ a\ \rightarrow\ Integer\\ cosa\ =\ redCola\ (+)\ (\lambda x\ \rightarrow\ 1)\end{array}$$

(D).– Prueba por inducción, $\forall\ c\ \centerdot\ c\ ::\ Cola\ a\ \centerdot\ 0\ <\ cosa\ c$

(E).– Deducir el tipo de la función definida por la ecuación

$$curiosa\ h\ =\ redCola\ (:>)\ (U\ .\ h)$$

¿Qué computa? Justifica la respuesta.

(F).– Utilizando la función $redCola$ define la función que *concatena* una cola de colas

$$concaCola\ ::\ Cola\ (Cola\ a)\ \rightarrow\ Cola\ a$$

Por ejemplo:

$$\begin{array}{l}\text{MAIN}>\ concaCola\\ \ \ (U\ (U\ 1\ :>\ U\ 2)\ :>\ U\ (U\ 3\ :>\ U\ 4)\ :>\ U\ (U\ 5\ :>\ U\ 6\ :>\ U\ 7))\\ (U\ 1\ :>\ U\ 2)\ :>\ (U\ 3\ :>\ U\ 4)\ :>\ U\ 5\ :>\ U\ 6\ :>\ U\ 7\ ::\ Cola\ Integer\end{array}$$

18.65 (PD: Febrero, 2000)

(A).– Descríbase una red de procesos, así como sus ecuaciones, para calcular la lista infinita de los elementos de la sucesión definida por

$$x_0 = 1, \qquad x_1 = 2, \qquad x_{n+2} = nx_{n+1} + x_n$$

(B).– Escribe una función para calcular el menor valor de n verificando

$$x_{n+2} - nx_n + x_{n+1} > 1000$$

18.66 (LabV: Febrero, 2000) (fichero: PD Febrero 00.hs) Sea la estructura de datos del Ejercicio 18.64.
(A).– Define las siguientes funciones:

```
fusion :: Cola a → Cola a → Cola a
  – – une dos colas
listaACola :: [a] → Cola a
  – – trasforma una lista en una cola equilibrada de forma que
  – – la primera mitad de la lista aparece en la subcola izquierda
```

Ayuda: utiliza la función *splitAt* de PRELUDE.
(B).– A partir de la función de *plegado* de colas del Ejercicio 18.64 definimos la función

```
pamCola x = redCola (:>) (U.($ x))
```

¿Cual es su tipo? ¿Qué hace? Justifica la respuesta con ejemplos.
(C).– Utilizando la función *redCola* define las siguientes funciones:

```
sumaCola :: Num a ⇒ Cola a → a
  – – la suma de los elementos de una cola
pertenece :: Eq a ⇒ a → Cola a → Bool
  – – el test de pertenencia
```

(D).– Sean ahora las siguientes estructuras para representar polinomios de coeficientes reales:

```
type Monomio  = (Integer, Double)   – – grado y coeficiente
type Polinomio = Cola Monomio       – – secuencia arbórea de monomios
```

donde supondremos que cada grado aparece una sola vez. Define las siguientes funciones, utilizando cuando se pueda la función *redCola*:

```
valorEn :: Double → Polinomio → Double
  – – valor de un polinomio en un punto
mulPorMonomio :: Monomio → Polinomio → Polinomio
  – – producto de monomio por polinomio
estáGrado :: Integer → Polinomio → Bool
  – – aparece el coef. de grado primer argumento
sumaUnMonomio :: Monomio → Polinomio → Polinomio
  – – suma un monomio a un polinomio
```

18.67 (PD: Extraordinario de Febrero, 2000) Sea la siguiente función de construcción de listas

$$\begin{array}{l} recons\ f\ z\ [\,] \quad\quad = [\,] \\ recons\ f\ z\ (x : xs) \ = \ z' : recons\ f\ z'\ xs\ \mathbf{where}\ z' \ = \ f\ z\ x \end{array}$$

(A).– Deduce su tipo.
(B).– Describe con un dibujo cómo actúa.
(C).– Define la función $separa$ que toma una lista no vacía y devuelve un par donde donde el segundo elemento es el último elemento de la lista, y el primero el resto de la lista

$$separa\ ::\ [a]\ \rightarrow\ ([a], a)$$

Por ejemplo, $separa\ [1,2,3,4] \implies ([1,2,3], 4)$
(D).– Prueba por inducción estructural que para listas no vacías se cumple la igualdad:

$$xs\ =\ (\ us +\!\!+ [y]\ \mathbf{where}\ (us, y)\ =\ separa\ xs\ )$$

(E).– Define, utilizando las funciones anteriores ($separa$ y $recons$), las funciones:

$sumar\ ::\ Num\ a\ \Rightarrow\ [a]\ \rightarrow\ a$
– – la suma de los elementos de la lista argumento
$máximo\ ::\ Ord\ a\ \Rightarrow\ [a]\ \rightarrow\ a$
– – el máximo de una lista no vacía

(F).– Si representamos un polinomio de coeficientes reales por la lista de sus coeficientes ($[Double]$), describe, utilizando las funciones anteriores ($separa$ y $recons$), una función

$$ruffini\ ::\ Double\ \rightarrow\ [Double]\ \rightarrow\ ([Double], Double)$$

para calcular el polinomio que resulta de dividir el polinomio segundo argumento por el monomio $x - u$ (donde u es el primer argumento) y el resto de la división.

18.68 (PD: Extraordinario de Febrero, 2000) Sea la siguiente función

$$\begin{array}{l} pam\ ::\ a\ \rightarrow\ [a\ \rightarrow\ b]\ \rightarrow\ [b] \\ pam\ x\ [\,] \quad\quad = [\,] \\ pam\ x\ (f : fs)\ =\ f\ x\ :\ pam\ x\ fs \end{array}$$

(A).– Define a partir de la función estandar map, una función pam' igual a la anterior.
(B).– Prueba por inducción estructural que efectivamente son iguales.

18.69 (LabIV: Junio, 2000) En un torneo de fútbol participan cuatro equipos teniendo que jugar todos contra todos. El resultado de cada partido puede ser "1" que significa que gana el primero de los dos equipos, "x" que significa que empatan y "2" que significa que gana el segundo equipo. Para representar la información se definen los siguiente tipos:

```
-- Nombre de equipos
type Equipo = String
-- Puntuación obtenida por un equipo
type Puntos = (Equipo, Int)
-- Resultado de un encuentro. Puede ser "1", "x" o "2"
type Resultado = String
-- Partido sin jugar. Es un par de equipos
type Partido = (Equipo, Equipo)
-- Resultado de un partido jugado.
--  "1" es que ganó el primero,
--  "x" que empataron y
--  "2" que ganó el segundo
type PartidoJugado = (Equipo, Equipo, Resultado)
-- Un torneo es una lista de partidos con sus resultados
type Torneo = [PartidoJugado]
```

Consideremos la lista de participantes en el torneo definida por:

```
países :: [Equipo] países = ["esp", "nor", "yug", "esl"]
```

(A).– Define una función *seJugó* :: $PartidoJugado \rightarrow Torneo \rightarrow Bool$ que determine si un partido $(e1, e2, r)$ se encuentra dentro de una lista de partidos teniendo en cuenta que por ejemplo ("nor", "esp", "1") puede aparecer en la lista como ("esp", "nor", "2"). Igual ocurre con los partidos empatados.

Define también una función *seJugaron* :: $[PartidoJugado] \rightarrow Torneo \rightarrow Bool$ que determine si todos los partidos de la lista primer argumento se jugaron en el torneo representado por la lista segundo argumento. Por ejemplo:

```
MAIN> seJugaron [("nor", "esp", "1"), ("yug", "esl", "x")]
          [("esp", "nor", "2"), ("esp", "yug", "1"), ("nor", "yug", "x"),
           ("esl", "esp", "x"), ("esl", "nor", "2"), ("esl", "yug", "x")]
True :: Bool
```

(B).– Define una función *enfrentamientos* :: $[Partido]$ que genere todos los posibles enfrentamientos entre los equipos participantes. Ten en cuenta que el enfrentamiento ("nor", "esp") es el mismo que el enfrentamiento ("esp", "nor"). Por ejemplo:

```
MAIN> enfrentamientos
[("esp", "nor"), ("esp", "yug"), ("nor", "yug"),
 ("esl", "esp"), ("esl", "nor"), ("esl", "yug")]
```

Define una función *juegaTorneo* :: $[Partido] \rightarrow [Resultado] \rightarrow Torneo$ en la que dada la lista de enfrentamientos del torneo y una lista de la misma longitud de resultados, empareje cada partido con su resultado correspondiente. Por ejemplo:

```
MAIN> juegaTorneo enfrentamientos ["1", "x", "x", "2", "1", "x"]
[("esp", "nor", "1"), ("esp", "yug", "x"), ("nor", "yug", "x"),
 ("esl", "esp", "2"), ("esl", "nor", "1"), ("esl", "yug", "x")]
```

(C).– Teniendo en cuenta que la función $columnas :: Int \rightarrow [[Resultado]]$ genera todos los posibles resultados para un número dado de partidos:

$$\begin{aligned} columnas\ 0 &= [[\,]] \\ columnas\ (n+1) &= map\ (\text{"1"} :)\ cs +\!\!+ map\ (\text{"x"} :)\ cs +\!\!+ map\ (\text{"2"} :)\ cs \\ &\quad \textbf{where}\ cs = columnas\ n \end{aligned}$$

define $resultados :: [Torneo]$ como una lista con todos los resultados posibles que se pueden dar en el torneo. Por ejemplo:

```
MAIN> resultados
[[("esp", "nor", "1"), ("esp", "yug", "1"), ("nor", "yug", "1"),
  ("esl", "esp", "1"), ("esl", "nor", "1"), ("esl", "yug", "1")],
 [("esp", "nor", "1"), ("esp", "yug", "1"), ("nor", "yug", "1"),
  ("esl", "esp", "1"), ("esl", "nor", "1"), ("esl", "yug", "x")],
 [("esp", "nor", "1"), ("esp", "yug", "1"), ("nor", "yug", "1"),
  ("esl", "esp", "1"), ("esl", "nor", "1"), ("esl", "yug", "2")],
...
 [("esp", "nor", "2"), ("esp", "yug", "2"), ("nor", "yug", "2"),
  ("esl", "esp", "2"), ("esl", "nor", "2"), ("esl", "yug", "2")]]
```

Define una función $puntos :: Torneo \rightarrow [Puntos]$ que proporcione la puntuación obtenida por cada equipo en un torneo dado. Si un equipo gana un partido, acumula 3 puntos, si empata 1 y nada si pierde. Por ejemplo:

```
MAIN> puntos [("yug", "esl", "x"), ("nor", "esl", "x"), ("nor", "yug", "2"),
              ("esp", "esl", "x"), ("esp", "yug", "1"), ("esp", "nor", "2")]
[("yug", 4), ("esl", 3), ("nor", 4), ("esp", 4)]
```

(D).– Define una función $ganadores :: [Puntos] \rightarrow ([Equipo], Int)$ que dada una lista de puntos para cada equipo, devuelva un par formado por la lista de equipo/s ganador/es y la puntuación obtenida. Por ejemplo:

```
MAIN> ganadores [("yug", 4), ("esl", 3), ("nor", 4), ("esp", 4)]
(["yug", "nor", "esp"], 4)
```

Define una función $posibilidad :: Equipo \rightarrow [PartidoJugado] \rightarrow [Torneo]$ tal que, dado un equipo y una lista de (algunos) partidos jugados de un torneo, devuelva los torneos completos a los que pueden pertenecer estos partidos y donde el equipo dado es ganador. Por ejemplo:

```
MAIN> posibilidad "esp" [("nor", "esp", "1"), ("yug", "esl", "x")]
[[("yug", "esl", "x"), ("nor", "esl", "1"), ("nor", "yug", "2"),
  ("esp", "esl", "1"), ("esp", "yug", "1"), ("esp", "nor", "2")],
 [("yug", "esl", "x"), ("nor", "esl", "x"), ("nor", "yug", "x"),
  ("esp", "esl", "1"), ("esp", "yug", "1"), ("esp", "nor", "2")],
 ...]
```

18.70 (PDI: Junio, 2000) **(A)**.– Dadas las definiciones siguientes:

$$\begin{array}{ll} [\,] +\!\!+ ys = ys & rev\,[\,] = [\,] \\ (x : xs) +\!\!+ ys = x : (xs +\!\!+ ys) & rev\,(x : xs) = rev\,xs +\!\!+ [x] \end{array}$$

y sabiendo que $+\!\!+$ es asociativo y que $[\,]$ es elemento neutro de $+\!\!+$ por la derecha, expresa cómo se aplica el principio de inducción estructural para demostrar la siguiente equivalencia:

$$\forall\, xs,\ ys :: [a] \,.\, rev\,(xs +\!\!+ ys) = rev\,ys +\!\!+ rev\,xs$$

y demuestra la equivalencia anterior.
(B).– Dada la definición de tipo

$$\mathbf{data}\ Htree\ a = H\ a \mid Htree\ a\ :\uparrow:\ Htree\ a$$

Define las funciones siguientes dando además los tipos correspondientes:

- *frontera* que aplicada un árbol del tipo anterior, produce la lista con el valor de sus hojas recorridas de izquierda a derecha.
- *revHtree* que aplicada un árbol del tipo anterior, produce otro árbol del mismo tipo, con las mismas hojas pero con la frontera invertida.

Define una función de plegado *foldHtree* (esquema recursivo) para el tipo *Htree* dando su tipo y expresa las funciones *frontera* y *revHtree* como aplicaciones de *foldHtree*

Construir una función *anivel* que sustituya el valor de cada hoja por el nivel en que se encuentra dicha hoja. Por ejemplo, *anivel* aplicado al árbol de la izquierda de la figura Figura 18.5, produce el de la derecha.
(C).– Genera una red y la expresión HASKELL que de la red se deduce para construir la lista de las sucesivas sumas de los naturales a partir del 1.

$$sumas = [1, (1+2), (1+2+3), (1+2+3+4),\ \ldots]$$

18.71 (PD: Extraordinario de Setiembre, 2000) Dadas las funciones

$$\begin{array}{l} rep\ x = x : x : rep\ x \\ tomar\ ys = head\,(tail\ ys) \end{array}$$

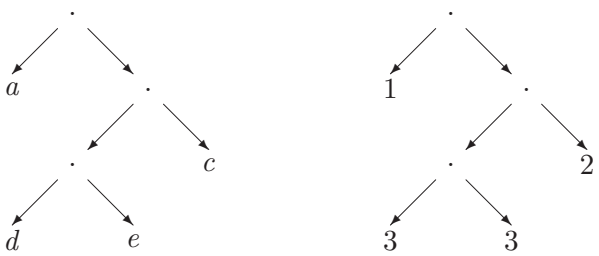

Figura 18.5: Resultado de la función $anivel$..

deduce sus tipos y reduce paso a paso la expresión $tomar\ (rep\ 1)$ usando evaluación perezosa.

18.72 (PD: Extraordinario de Setiembre, 2000) Dada la siguiente definición de árbol general:

$$\mathbf{data}\ ArbGen\ a\ =\ NodoG\ a\ [ArbGen\ a]$$

y la función de plegado

$$\begin{array}{l} foldAG\ ::\ (a \to [b] \to\ b)\ \to\ ArbGen\ a\ \to\ b \\ foldAG\ f\ (Nodo\ a\ xs)\ =\ f\ a\ (map\ (foldAG\ f)\ xs) \end{array}$$

(A).– Utiliza dicha función para definir una función $nodos$ que calcule el número de nodos de un árbol dado dando su tipo.
(B).– Utiliza dicha función para definir una función $preorden$ que calcule un recorrido en preorden del árbol dado dando su tipo.

18.73 (PD: Extraordinario de Setiembre, 2000) Describe una red y la expresión HASKELL que de la red se deduce, para cada una de las expresiones siguientes:

$$\begin{array}{lcl} factI & = & [1!, 3!, 5!, \ldots] \\ pot\ x & = & [x, -x^3, x^5, -x^7, \ldots] \\ coef\ x & = & [x, -x^3/3!, x^5/5!, -x^7/7!, \ldots] \end{array}$$

18.74 (LabV: Extraordinario de Setiembre, 2000) (fichero: LabV Setiembre 00.hs) Para calcular el número de diputados que obtienen distintos partidos políticos en unas elecciones se definen los siguientes tipos:

$$\begin{array}{lcl} \mathbf{type}\ NombrePartido & = & String \\ \mathbf{type}\ NumeroDeDiputados & = & Int \\ \mathbf{type}\ NumeroDeVotos & = & Int \end{array}$$

$\textbf{data}\ Partido\ a\ =\ P\ NombrePartido\ a\ \textbf{deriving}\ Show$

$\textbf{type}\ PartidoYVotos\ =\ Partido\ NumeroDeVotos$
– – nombre del partido con los votos obtenidos
$\textbf{type}\ PartidoYDiputados\ =\ Partido\ NumeroDeDiputados$
– – nombre del partido con los diputados obtenidos

(A).– Construye una función

$$insertaV\ ::\ PartidoYVotos\ \rightarrow\ [PartidoYVotos]\ \rightarrow\ [PartidoYVotos]$$

que inserte un partido con sus votos en unas elecciones (listas de partidos con votos). Supón que el partido que estamos insertando no se encuentra ya incluido en las elecciones y que la lista segundo argumento se encuentra ordenada por los votos de mayor a menor. La inserción debe hacerse también ordenada por el número de votos, de manera que el partido más votado aparezca en la cabeza de la lista. Por ejemplo:

$$\begin{array}{l}\text{MAIN}>\ insertaV\ (P\ "PSOE"\ 21000)\\ \qquad\qquad\qquad [P\ "PP"\ 11000,\ P\ "IU"\ 8000]\end{array}$$

$$[P\ "PSOE"\ 21000, P\ "PP"\ 11000, P\ "IU"\ 8000]\ ::\ [PartidoYVotos]$$

(B).– Define la función $ordenaV$ que ordene una lista de partidos con sus votos:

$$ordenaV\ ::\ [PartidoYVotos]\ \rightarrow\ [PartidoYVotos]$$

Por ejemplo:

$$\begin{array}{l}\text{MAIN}>\ ordenaV\ [P\ "IU"\ 16000,\ P\ "PSOE"\ 21000,\ P\ "PP"\ 22000]\\ [P\ "PP"\ 22000, P\ "PSOE"\ 21000, P\ "IU"\ 16000]\ ::\ [PartidoYVotos]\end{array}$$

(C).– Define una función

$$sumaV\ ::\ PartidoYVotos\ \rightarrow\ [PartidoYVotos]\ \rightarrow\ [PartidoYVotos]$$

que acumule los datos obtenidos por un partido (por ejemplo en una provincia) a los de unas elecciones. Supón que los partidos en las elecciones están ordenados y devuelve el resultado ordenado. Por ejemplo:

$$\begin{array}{l}\text{MAIN}>\ sumaV\ (P\ "PSOE"\ 21000)\\ \qquad\qquad\qquad [P\ "PSOE"\ 17000,\ P\ "PP"\ 11000,\ P\ "IU"\ 8000]\end{array}$$

$$[P\ "PSOE"\ 38000, P\ "PP"\ 11000, P\ "IU"\ 8000]\ ::\ [PartidoYVotos]$$

NOTA: El partido que vamos a acumular puede que aún no aparezca en las elecciones.
(D).– Define una función

$$juntarV \;::\; [PartidoYVotos] \;\rightarrow\; [PartidoYVotos] \;\rightarrow\; [PartidoYVotos]$$

que sume los datos obtenidos por los partidos en las elecciones en dos provincias (Suponer los argumentos ordenados y el resultado también debe estar ordenado). Por ejemplo:

$$\begin{array}{l}\text{MAIN}> \; juntarV\; [P\text{ "}PSOE\text{" }17000,\; P\text{ "}PP\text{" }11000,\; P\text{ "}IU\text{" }8000] \\ \qquad\qquad\qquad\quad [P\text{ "}PSOE\text{" }10000,\; P\text{ "}IU\text{" }8000,\; P\text{ "}PP\text{" }200\,]\end{array}$$

$$[P\text{ "}PSOE\text{" }27000, P\text{ "}IU\text{" }16000, P\text{ "}PP\text{" }11200] \;::\; [PartidoYVotos]$$

(E).– La ley para el cálculo del número de diputados que corresponde a cada partido una vez conocido el resultado de las elecciones y el número de diputados a repartir es la siguiente[1]:

> *Mientras queden diputados por asignar, se toma el partido de mayor número de votos, y se le da un candidato. A continuación, se considera a ese partido con la mitad de votos que tenía en ese momento y se repite el proceso.*

Como resultado obtendremos una lista de tipo $[PartidoYDiputados]$ con la asignación de candidatos por partido. Escribir la función $d'Hont$ que calcule los diputados de cada partido

$$d'Hont \;::\; NumeroDeDiputados \;\rightarrow\; [PartidoYVotos] \;\rightarrow\; [PartidoYDiputados]$$

Por ejemplo, para la lista $[P\text{ "}PSOE\text{" }17000,\; P\text{ "}PP\text{" }11000,\; P\text{ "}IU\text{" }8000]$, y 6 diputados (los subíndices indican el orden de asignación de los candidatos), tenemos:

Partido	Votos	Diput.	V/2	Diput.	V/2	Diput.	V/2	Total Dip.
PSOE	17000_1	+1	8500_3	+1	4250_6	+1	2125	3
PP	11000_2	+1	5500_5	+1	2750			2
IU	8000_4	+1	4000					1

En este caso,

$$\begin{array}{l}\text{MAIN}> \; d'Hont\; 6\; [P\text{ "}PSOE\text{" }17000,\; P\text{ "}PP\text{" }11000,\; P\text{ "}IU\text{" }8000] \\ [P\text{ "}PSOE\text{" }3, P\text{ "}PP\text{" }2, P\text{ "}IU\text{" }1] \;::\; [PartidoYDiputados]\end{array}$$

$$\begin{array}{l}\text{MAIN}> \; d'Hont\; 5\; [P\text{ "}PSOE\text{" }17000,\; P\text{ "}PP\text{" }11100,\; P\text{ "}IU\text{" }11000] \\ [P\text{ "}PP\text{" }2, P\text{ "}PSOE\text{" }2, P\text{ "}IU\text{" }1] \;::\; [PartidoYDiputados]\end{array}$$

$$\begin{array}{l}\text{MAIN}> \; d'Hont\; 8\; [P\text{ "}PSOE\text{" }17000,\; P\text{ "}PP\text{" }11000,\; P\text{ "}IU\text{" }8000] \\ [P\text{ "}PP\text{" }3, P\text{ "}PSOE\text{" }3, P\text{ "}IU\text{" }2] \;::\; [PartidoYDiputados]\end{array}$$

[1] Realmente se trata de una simplificación de la conocida ley d'Hont.