19 SOLUCIONES A LOS EJERCICIOS DE EXÁMENES

19.1. AÑO 1996

Solución al Ejercicio 18.1 (pág. 513).– **(A)**.–

$$h\ p\ =\ foldr\ (\lambda\ x\ u\ \rightarrow\ \textbf{if}\ p\ x\ \textbf{then}\ x : u\ \textbf{else}\ u)\ [\,]$$

(B).– Una posible función, sin utilizar la función h

$$\begin{array}{l} divisores\ x\ =\ [\ y\ |\ y \leftarrow [a, a-1..(-a)],\ y \neq\ 0,\ x\ `mod`\ y == 0] \\ \quad \textbf{where}\ a = abs\ x \end{array}$$

y otra solución utilizando h

$$\begin{array}{l} divisores'\ x\ =\ h\ (\lambda\ y\ \rightarrow\ y \neq\ 0\ \&\&\ x\ `mod`\ y == 0)\ [a, a-1..(-a)] \\ \quad \textbf{where}\ a = abs\ x \end{array}$$

Obsérvese que se obtiene la lista de divisores en orden creciente.
(C).– En principio, consideremos una función para evaluar un polinomio en un punto, utilizando la regla de Ruffini

$$val\ x\ =\ foldr\ (\lambda\ a\ b\ \rightarrow\ a + x * b)\ 0$$

Entonces, ya que para el polinomio $xs@(a0 : \_)$, $a0$ es el término independiente, hay que calcular la lista de posibles candidatos a raíces, que será

$$h\ (\lambda\ x\ \rightarrow\ val\ x\ xs == 0)(divisores\ a0)$$

y después tomar la cabeza (si ésta no es vacía)

$$\begin{array}{l} mayorRaízEntera\ xs@(a0 : \_)\ =\ \textbf{case}\ ps\ \textbf{of}\ [\,] \quad\quad \rightarrow\ NoExiste \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad (x : \_) \rightarrow Raíz\ x \\ \quad \textbf{where} \\ \qquad ps\ =\ h\ (\lambda\ x\ \rightarrow\ val\ x\ xs == 0)(divisores\ a0) \end{array}$$

Solución al Ejercicio 18.2 (pág. 513).– Una función polimórfica puede ser la identidad

$$\begin{array}{l} id \;::\; a \;\to\; a \\ id\; x \;=\; x \end{array}$$

mientras que una sobrecargada podría ser, por ejemplo, la función $negate$ de la clase Num:

$$\begin{array}{l} \mathbf{class}\;(Eq\;a,\;Show\;a) \;\Rightarrow\; Num\;a\;\mathbf{where} \\ \quad (+),\;(-),\;(*) \;::\; a \;\to\; a \;\to\; a \\ \quad negate \qquad\quad ::\; a \;\to\; a \\ \quad \ldots \end{array}$$

Solución al Ejercicio 18.3 (pág. 513).– **(A)**.– Si tomamos $h \equiv \lambda\, g \;\to\; g\; x$, entonces tenemos la ecuación

$$h\; g \;=\; g\; x$$

y asignando tipos al argumento ($g \;::\; \alpha$) y al resultado ($g\; x \;::\; \delta$), tenemos

$$(g\; x \;::\; \delta),\; x :: \beta$$

de donde $g \;::\; \beta \;\to\; \delta$, y de aquí $h \;::\; (\beta \;\to\; \delta) \;\to\; \delta$.
(B).– Asignando de nuevo tipos a los argumentos ($f \;::\; \alpha,\; x :: \beta$) y también al resultado ($f\;(\lambda\, g \;\to\; g\; x) :: \pi$) tenemos

$$f\;(\lambda\, g \;\to\; g\; x) \;::\; \pi, \qquad (\lambda\, g \;\to\; g\; x) \;::\; (\beta \;\to\; \delta) \to \delta$$

de donde $f \;::\; ((\beta \to \delta) \to \delta) \;\to\; \pi$, y de aquí

$$pen \;::\; (((\beta \;\to\; \delta) \to \delta) \;\to\; \pi) \to \beta \;\to\; \pi$$

Solución al Ejercicio 18.4 (pág. 514).– El esquema de inducción es

$$\begin{array}{ll} & \forall\, x \;\centerdot\; x :: Ent \;\centerdot\; O - (O - x) = x \\ \equiv & \{\,\text{esquema de inducción}\,\} \\ & O - (O - O) \;=\; O \quad \text{-- Caso Base} \\ \wedge & \\ & \forall\, x \;\centerdot\; x :: Ent \;\centerdot\; O - (O - x) = x \\ \Rightarrow & \{\;\textit{Paso Inductivo}\,\} \\ & (O - (O - P\; x) = P\; x \;\wedge\; O - (O - S\; x) = S\; x \end{array}$$

El caso base es trivial:

$$\begin{array}{ll} & O - (O - O) \\ \equiv & \{\,1)\text{-}\,\} \\ & O - O \\ \equiv & \{\,1)\text{-}\,\} \end{array}$$

$$O$$

Y el paso inductivo constará de dos fases, una para cada constructor:

$$\begin{array}{ll} & O - (O - P\ x) \\ \equiv & \{\,(3)\text{-}\,\} \\ & O - S\ (O - x) \\ \equiv & \{\,2)\text{-}\,\} \\ & P(O - (O - x)) \\ \equiv & \{\ \text{hipótesis de inducción}\,\} \\ & P\ x \end{array}$$

$$\begin{array}{ll} & O - (O - S\ x) \\ \equiv & \{\,2)\text{-}\,\} \\ & O - P(O - x) \\ \equiv & \{\,(3)\text{-}\,\} \\ & S(O - (O - x)) \\ \equiv & \{\ \text{hipótesis de inducción}\,\} \\ & S\ x \end{array}$$

Solución al Ejercicio 18.5 (pág. 514).– Hay que demostrar,

$$\forall\, x, xs\ \centerdot\ x :: a,\ xs :: [a]\ \centerdot\ (f.head)\ (x : xs)\ = (head.map\ f)\ (x : xs)$$

$$\begin{array}{ll} & (f.head)\ (x : xs)\ =\ (head.map\ f)\ (x : xs) \\ \equiv & \{\,\text{definición composición}\,\} \\ & f\ (head\ (x : xs))\ =\ head\ (map\ f\ (x : xs)) \\ \equiv & \{\,\text{definición } head \text{ y } map\,\} \\ & f\ x\ =\ head\ (f\ x\ :\ map\ f\ xs) \\ \equiv & \{\,\text{definición } head\,\} \\ & f\ x\ =\ f\ x \end{array}$$

Obsérvese que la demostración es directa, sin utilizar la hipótesis de inducción.

Solución al Ejercicio 18.6 (pág. 514).– **(A)**.–

$$\mathbf{data}\ Ater\ a\ =\ Vacío \mid Nodo\ a\ (Ater\ a)\ (Ater\ a)\ (Ater\ a)\ \mathbf{deriving}\ Show$$

(B).–

$$\begin{array}{lll} nNodos\ Vacío & = & 0 \\ nNodos\ (Nodo\ x\ i\ c\ d) & = & 1\ +\ nNodos\ i\ +\ nNodos\ c\ +\ nNodos\ d \end{array}$$

(C).–

$$\begin{array}{lll} pliegat\ \_\ z\ Vacío & = & z \\ pliegat\ f\ z\ (Nodo\ x\ i\ c\ d) & = & f\ x\ (pliegat\ f\ z\ i)\ (pliegat\ f\ z\ c)(pliegat\ f\ z\ d) \end{array}$$

(D).–

$$nNodos \;=\; pliegat\;(\lambda\_\; ni\; nc\; nd \;\rightarrow\; 1 + ni + nc + nd)\; 0$$

Solución al Ejercicio 18.7 (pág. 514).– **(A)**.– Obsérvese que en cada lista la suma es constante: $2, 3, \ldots$. Una posible solución directa es:

$$pares \;=\; [\,[(s - x, x) | x \leftarrow [1..s-1]\,] \mid s \leftarrow [2..]\,]$$

Otra solución se obtiene observando que, a partir de una lista, la siguiente se obtiene al sumar a la segunda componente la unidad para cada elemento de la lista anterior:

$$\begin{array}{l} pares \;=\; [(1,1)] : \; map\; f\; pares \\ f\;(ps@(x,y) : \_) \;=\; (x+1, y) \;:\; map\;(\lambda\,(u,v) \rightarrow (u, v+1))\; ps \end{array}$$

Solución al Ejercicio 18.8 (pág. 514).– **(A)**.–

$$test\; a\; b\; c\;(u,v) \;=\; a + b + c == u * 10 + v$$

$$\begin{array}{l} suma\; a\; b\; c \;=\; (u,v) \\ \quad \textbf{where}\; s \;=\; a + b + c \\ \qquad\quad\; u \;=\; s\; `div`\; 10 \\ \qquad\quad\; v \;=\; s\; `mod`\; 10 \end{array}$$

(B).–

$$\begin{array}{l} sumar\;[\,]\;\;\;\;\;\;[\,]\;\;\;\;\;\;\;\;= (0, [\,]) \\ sumar\;(x : xs)\;(y : ys) \;=\; (u, v : zs) \\ \quad \textbf{where}\;(ac, zs) \;=\; sumar\; xs\; ys \\ \qquad\quad\;\; (u, v) \;\;\;=\; suma\; ac\; x\; y \\ sumaListas\; xs\; ys \;=\; \textbf{if}\;\; a == 0\;\; \textbf{then}\;\; ss\;\; \textbf{else}\;\; a : ss \\ \quad\;\; \textbf{where}\;(a, ss) \;=\; sumar\; xs\; ys \end{array}$$

(C).– Solamente hay que añadir la línea

$$\ldots,\; \textbf{let}\;(ac'', m') \;=\; suma\; ac'\; a\; a,\; m == m',\; ac'' == 0\,]$$

Las soluciones vienen dadas por el siguiente diálogo

$$\begin{array}{l} \text{MAIN}> \; sol \\ [[(1,8,1),(1,3,1),(3,1,2)],[(2,7,2),(2,5,2),(5,2,4)], \\ [(3,6,3),(3,7,3),(7,3,6)]\;,[(4,5,4),(4,9,4),(9,4,8)]] \;::\; [[(Int, Int, Int)]] \end{array}$$

Obsérvese que en la tercera solución:

$$\begin{array}{rrrr} & 3 & 6 & 3 \\ + & 3 & 7 & 3 \\ \hline & 7 & 3 & 6 \end{array} \qquad\qquad \begin{array}{rrrr} & a & n & a \\ + & a & m & a \\ \hline & m & a & s \end{array}$$

las cifras asociadas a n y a s coinciden con 6.

(D).– Si no se permiten repeticiones basta con añadir guardas cada vez que aparezca una nueva cifra:

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

```
MAIN> sol'
[[(1,8,1),(1,3,1),(3,1,2)],
[(2,7,2),(2,5,2),(5,2,4)],
[(4,5,4),(4,9,4),(9,4,8)]] :: [[(Int,Int,Int)]
```

y ha desaparecido la solución con cifras repetidas. Para obtener un conjunto de dígitos que proporcione una única solución basta tomar el conjunto de dígitos de cualquiera de las soluciones; por ejemplo, tomando $ds = [1,3,2,8]$, la solución será única.

Solución al Ejercicio 18.9 (pág. 515).– Ver también Ejercicio 18.8.

(A).–

```
resta ac b a = if s ≤ a then (0, a − s) else (1, a + 10 − s)
  where s = ac + b
test a b c (u,v) = 10 * u + c − (a + b)== v
```

(B).–

```
mayor []     []     = False
mayor (x:xs) (y:ys) = if x== y then
                        mayor xs ys
                      else
                        x < y
restaListas []     []     = (0,[])
restaListas (x:xs) (y:ys) = (u, v:zs)
  where
    (ac,zs) = restaListas xs ys
    (u,v)   = resta ac x y
```

(C).– Hay que añadir la siguiente línea

```
..., let (ac'',a'') = resta ac' a m, a''== a, ac''== 0 ]
```

(D).–

$$
\begin{array}{l}
solr \;=\; [\,[(m,a,s),(a,n,a),(a,m,a)]\mid \\
\qquad\quad s \leftarrow\; ds, \\
\qquad\quad a \leftarrow\; ds,\; s \neq\; a, \\
\qquad\quad \textbf{let}\;(ac,a') = resta\;0\;a\;s,\; a'\texttt{==}\;a, \\
\qquad\quad n \leftarrow\; ds,\; n \neq\; a,\; n \neq\; s, \\
\qquad\quad \textbf{let}\;(ac',m) \;=\; resta\;ac\;n\;a,\; m \neq\; n,\; m \neq\; a,\; m \neq\; s, \\
\qquad\quad \textbf{let}\;(ac'',a'') = resta\;ac'\;a\;m, a''\texttt{==}\;a, ac''\texttt{==}\;0\,]
\end{array}
$$

Para obtener un conjunto de dígitos que proporcione una única solución basta tomar el conjunto de dígitos de cualquiera de las soluciones.

Solución al Ejercicio 18.10 (pág. 516).– **(A)**.– Ya que

$$
\begin{array}{l}
(.) \quad :: (b \;\to\; c) \;\to\; (a \;\to\; b) \;\to\; (a \;\to\; c) \\
map \;::\; (u \;\to\; v\,) \;\to\; [u] \;\to\; [v]
\end{array}
$$

si fuera $f \;::\; u \;\to\; v$, entonces $map\,f \;::\; [u] \;\to\; [v]$, e identificando se obtiene:

$$[u] \;\to\; [v] \;\equiv\; (b \;\to\; c) \;\equiv\; (a \;\to\; b)$$

y de aquí, $a \equiv\; b \equiv\; c \equiv\; [u],\; u \equiv\; v$, y por tanto $g \;::\; [u] \;\to\; [u]$.
(B).– Demostraremos por inducción estructural sobre la lista

$$
\begin{array}{l}
\quad \forall\, xs \;\centerdot\; xs \;::\; [a]\,(\forall\, f \;\centerdot\; f \;::\; a \to\; b \;\centerdot\; (map\,f\;.\;map\,f\,)\;xs \;=\; map\;(f.f)\;xs\,) \\
\equiv\equiv \\
\quad \forall\, xs \;\centerdot\; xs \;::\; [a]\,(\forall\, f \;\centerdot\; f \;::\; a \to\; b \;\centerdot\; map\,f\;(map\,f\;xs) \;=\; map\;(f.f)\;xs)
\end{array}
$$

—*Caso Base*:

$$
\begin{array}{ll}
 & map\,f\;(map\,f\;[\,]) \;=\; map\;(f.f)\;[\,] \\
\equiv\equiv & \{\,map\;\_\;[\,] = [\,]\,\} \\
 & map\,f\;[\,] \;=\; [\,] \\
\equiv\equiv & \{\,map\;\_\;[\,] = [\,]\,\} \\
 & Cierto
\end{array}
$$

—*Paso Inductivo*:

$$
\begin{array}{ll}
 & map\,f\;(map\,f\;(x:xs)) \;=\; map\;(f.f)\;(x:xs) \\
\equiv\equiv & \{\,map\;g\;(u:us) = g\;u : map\;g\;us\,\} \\
 & map\,f\;(f\;x \;:\; map\,f\;xs) \;=\; (f.f)x \;:\; map\;(f.f)\;xs \\
\equiv\equiv & \{\,\text{id. Anterior}\,\} \\
 & f\;(f\;x) \;:\; map\,f\;(map\,f\;xs) \;=\; (f.f)x \;:\; map\;(f.f)\;xs \\
\equiv\equiv & \{\,\text{definición de composición}\,\} \\
 & f\;(f\;x) \;:\; map\,f\;(map\,f\;xs) \;=\; f(f\;x) \;:\; map\;(f.f)\;xs \\
\Leftarrow & \{\;\text{la igualdad es sustitutiva}\,\} \\
 & map\,f\;(map\,f\;xs) \;=\; map\;(f.f)\;xs \\
\Leftarrow & \{\;\text{hipótesis de inducción}\,\} \\
 & Cierto
\end{array}
$$

Solución al Ejercicio 18.11 (pág. 516).– **(A)**.–

$$
\begin{array}{l}
\mathit{estáYresto}\ x\ [\,] \quad\quad = (\mathit{False},\ [\,]) \\
\mathit{estáYresto}\ x\ (y : ys) \\
\quad \mid x == y \quad\quad = (\mathit{True},\ ys) \\
\quad \mid \mathit{otherwise} \quad = (e,\ y : ys') \\
\quad \textbf{where} \\
\quad\quad (e, ys') \ = \ \mathit{estáYresto}\ x\ ys
\end{array}
$$

(B).–

$$
\begin{array}{l}
\mathit{contenidoYresto}\ [\,] \quad\quad ys \ = \ (\mathit{True}, ys) \\
\mathit{contenidoYresto}\ (x : xs) ys \ = \ (e\ \&\&\ f, rs') \\
\quad \textbf{where}\ (e, rs) \ = \ \mathit{estáYresto}\ x\ ys \\
\quad\quad\quad\quad\ (f, rs') \ = \ \mathit{contenidoYresto}\ xs\ rs
\end{array}
$$

Solución al Ejercicio 18.12 (pág. 517).– **(A)**.– h calcula el conjunto potencia.
(B).–

$$
\begin{array}{l}
\mathit{divisores}\ x \ = \ x \ : \ [\ y \mid y \leftarrow [1..x\ `\mathit{mod}`\ 2],\ x\ `\mathit{mod}`\ y == 0] \\
\mathit{perfecto}\ x \ = \ \mathit{foldr}\ (+)\ 0\ (\mathit{divisores}\ x\ ) == 2 * x
\end{array}
$$

Solución al Ejercicio 18.13 (pág. 517).– **(A)**.– Ver Ejercicio 18.4.
(B).–

$$
\begin{array}{l}
\mathit{suma} \quad\quad = \mathit{foldr}\ (+)\ O \\
\mathit{pertenece}\ z \ = \ \mathit{foldr}\ (\lambda\, x\ u \ \rightarrow \ x == z \ \|\ u)\ \mathit{False}
\end{array}
$$

(C).–

$$
\begin{array}{l}
\mathit{aEnt}\ x \\
\quad \mid \ x == 0 \quad\quad = O \\
\quad \mid \ x > 0 \quad\quad = S\ (\mathit{aEnt}(x - 1)) \\
\quad \mid \ \mathit{otherwise} \ = P\ (\mathit{aEnt}(x + 1)) \\
\\
\mathit{aInt}\ O \quad\quad = 0 \\
\mathit{aInt}\ (S\ x) \ = \ \mathit{aInt}\ x \ + \ 1 \\
\mathit{aInt}\ (P\ x) \ = \ \mathit{aInt}\ x - 1 \\
\\
\mathit{máximo}\ x\ y \ = \ \mathit{aEnt}\ (\mathit{max}\ (\mathit{aInt}\ x)\ (\mathit{aInt}\ y)\ )
\end{array}
$$

Solución al Ejercicio 18.14 (pág. 518).–

$$
\mathit{lista} \ = \ [1..] \ : \ \mathit{map}\ (\mathit{zipWith}\ (*)\ [1..])\ \mathit{lista}
$$

```
toma                  :: [[Int]] → [Int]
toma ((x : _) : ys)   =  x : toma (map tail ys)

sol = toma (tail lista)
```

Solución al Ejercicio 18.15 (pág. 518).– **(A)**.–

```
comunes [] _          = []
comunes (x : xs) ys
  | x `elem` ys       = x : comunes xs ys
  | otherwise         = comunes xs ys
```

Otra forma puede ser

```
comunes xs ys = foldr f [] xs
  where f x rs = if x `elem` ys then x : rs else rs
```

(B).–

```
pasa 0 xs bs          = (xs, bs)
pasa n xs (b : bs)
  | b `elem` xs       = pasa n xs bs
  | otherwise         = pasa (n − 1) (b : xs) bs
```

(C).–

```
incluye 0 (a : ac) = (a + 1) : ac
incluye n (a : ac) = a : incluye (n − 1) ac
```

Solución al Ejercicio 18.16 (pág. 519).– **(A)**.–

```
enCola x (Ult y)    = Ult x :> y
enCola x (c :> y)   = (enCola x c) :> y

deCola (c :> x) = (c, x)

mapCola f (Ult x)   = Ult (f x)
mapCola f (c :> x)  = (mapCola f c) :> f x
```

Para demostrar la igualdad que se pide utilizamos inducción

```
∀ c . c :: Cola a . ∀ x, f . x : a, f :: a → b .
      mapCola f (enCola x c) = enCola (f x) (mapCola f c)
≡≡ { principio de inducción }
   -- Caso Base:
   ∀ f, x, y . f :: a → b, x :: a, y :: a .
      mapCola f (enCola x (Ult y)) = enCola (f x) (mapCola f (Ult y))
∧
```

– – *Paso Inductivo*:

$$\begin{array}{l}\forall\ c\ \centerdot\ c :: Cola\ a\ \centerdot \\ \quad \forall f,\ x\ \centerdot\ f :: a \to b,\ x\ ::\ a\ \centerdot \\ \qquad mapCola\ f\ (\ enCola\ x\ c) \ =\ enCola\ (f\ x)\ (mapCola\ f\ c) \\ \Rightarrow \\ \quad \forall f, x, y\ \centerdot\ f :: a \to b,\ x :: a,\ y :: a\ \centerdot \\ \qquad mapCola\ f\ (enCola\ x\ (c :>\ y)) \ =\ enCola\ (f\ x)\ (mapCola\ f\ (c :>\ y))\end{array}$$

—*Caso Base*:

$$\begin{array}{ll} & mapCola\ f\ (enCola\ x\ (Ult\ y)) \ =\ enCola\ (f\ x)\ (mapCola\ f\ (Ult\ y)) \\ \equiv & \{\ 1)enCola,\ 1)mapCola\ \} \\ & mapCola\ f\ (Ult\ x\ :>\ y) \ =\ enCola\ (f\ x)\ (Ult\ (f\ y)) \\ \equiv & \{\ 2)mapCola,\ 1)enCola\ \} \\ & mapCola\ f\ (Ult\ x)\ :>\ f\ y \ =\ Ult\ (f\ x)\ :>\ f\ y \\ \equiv & \{\ 2)mapCola\ \} \\ & Ult\ (f\ x)\ :>\ f\ y \ =\ Ult\ (f\ x)\ :>\ f\ y \\ \equiv & \\ & Cierto\end{array}$$

—*Paso Inductivo*:

$$\begin{array}{ll} & mapCola\ f\ (enCola\ x\ (c :>\ y)) \ =\ enCola\ (f\ x)\ (mapCola\ f\ (c :>\ y)) \\ \equiv & \{\ 2)enCola,\ 2)mapCola\ \} \\ & mapCola\ f\ ((enCola\ x\ c)\ :>\ y) \ =\ enCola\ (f\ x)\ (\ (mapCola\ f\ c)\ :>\ f\ y) \\ \equiv & \{\ 2)mapCola,\ 2)enCola\ \} \\ & mapCola\ f\ (enCola\ x\ c)\ :>\ f\ y \ =\ (\ enCola\ (f\ x)\ (mapCola\ f\ c)\ )\ :>\ f\ y \\ \Leftarrow & \{\ is\ \} \\ & mapCola\ f\ (enCola\ x\ c) \ =\ enCola\ (f\ x)\ (mapCola\ f\ c) \\ \equiv & \{\ \text{hipótesis de inducción}\ \} \\ & Cierto\end{array}$$

(B).– Utilizando el esquema de reducción de colas $redCola$ del enunciado del ejercicio podemos escribir

$$\begin{array}{l} enCola\ x\ c \ =\ redCola\ (\ flip\ (:>)\ )\ (\lambda\ y\ \to\ (Ult\ x)\ :>\ y)\ c \\ \text{– – o también, de forma más compacta:} \\ enCola \ =\ (\ redCola\ .\ flip\ )\ (:>)\ .\ (:>)\ .\ Ult \end{array}$$

$$\begin{array}{l} colaALista\ ::\ Cola\ a\ \to\ [a] \\ colaALista\ =\ redCola\ (:)\ (:\ [\,]) \end{array}$$

(C).–

$$\begin{array}{lll} inversa & = & redCola\ enCola\ Ult \\ fusión & = & (redCola\ .\ flip)\ (:>)\ .\ (:>) \\ pertenece\ x & = & redCola\ ((||).\ (x==\ )\ )\ (x==\ ) \end{array}$$

(D).–

$$listaACola\ (x : xs) \ = \ foldl\ (flip\ enCola)\ (Ult\ x)\ xs$$

(E).–

$$mezcla \ = \ redCola\ (>>>)\ .\ (flip\ (>>>))$$

$$\begin{array}{l} y\ >>>\ (Ult\ x) \\ \quad |\ x\ >=\ y \qquad = Ult\ x\ :>\ y \\ \quad |\ otherwise \qquad = Ult\ y\ :>\ x \\ y\ >>>\ (c\ :>\ x) \\ \quad |\ x\ >=\ y \qquad = (c\ :>\ x)\ :>\ y \\ \quad |\ otherwise \qquad = (y\ >>>\ c)\ :>\ x \end{array}$$

$$ordena \ = \ redCola\ (mezcla\ .\ Ult\ )\ Ult$$

(F).–

$$separar\ c \ = \ (p, i)\ \mathbf{where}\ (\_, p, i) \ = \ separar'\ c$$

$$\begin{array}{l} separar'\ (Ult\ x\ :>\ y\ ) \qquad\qquad = (0, Ult\ x,\ Ult\ y) \\ separar'\ ((Ult\ x\ :>\ y)\ :>\ z\ ) \ = (1, Ult\ x\ :>\ z,\ Ult\ y) \\ separar'\ ((c\ :>\ y)\ :>\ z\ ) \\ \quad |\ i == 0 \qquad\qquad = (0, p\ :>\ y,\ p'\ :>\ z) \\ \quad |\ otherwise \qquad\qquad = (1, p\ :>\ z,\ p'\ :>\ y) \\ \quad\ \mathbf{where}\ (i,\ p, p') \ = \ separar'\ c \end{array}$$

19.2. AÑO 1997

Solución al Ejercicio 18.17 (pág. 520).– **(A).–**

$$\begin{array}{l} mapUrna\ f\ V \qquad = \ V \\ mapUrna\ f\ (U\ x\ u) \ = \ U\ (f\ x)\ (mapUrna\ f\ u) \end{array}$$

(B).–

$$\begin{array}{l} mapUrna' \ :: \ (a \ \to \ b) \ \to \ Urna\ a \ \to \ Urna\ b \\ mapUrna' f \ = \ pl\ (\lambda\, x\ u \ \to \ U\ (f\ x)\ u)\ V \end{array}$$

(C).–

$$\begin{array}{l} \forall\, u\ \centerdot\ u :: Urna\ a\ \centerdot\ (\forall\, f\ \centerdot\ f \ :: \ a \ \to \ b\ \centerdot\ mapUrna\ f\ u \ = \ mapUrna'\ f\ u) \\ \equiv \ \{\text{principio de inducción}\} \\ \quad \forall\, f\ \centerdot\ f \ :: \ a \ \to \ b\ \centerdot\ mapUrna\ f\ V \ = \ mapUrna'\ f\ V \quad \textit{-- Caso Base} \\ \wedge \\ \quad \forall\, u\ \centerdot\ x\ u :: Urna\ a,\ x :: a\ \centerdot\ \forall\, f\ \centerdot\ f \ :: \ a \ \to \ b\ \centerdot \\ \qquad\quad mapUrna\ f\ u \ = \ mapUrna'\ f\ u \end{array}$$

$\Rightarrow$ *– – Paso Inductivo*
$mapUrna\ f\ (U\ x\ u)\ =\ mapUrna'\ f\ (U\ x\ u)$

(D).– *Caso Base*

$$
\begin{array}{ll}
 & mapUrna\ f\ V\ =\ mapUrna'\ f\ V \\
\equiv & \{\ 1)mapUrna,\ 1)mapUrna'\ \} \\
 & V\ =\ pl\ (\ldots)\ V \\
\equiv & \{\ 1)\text{-}pl\ \} \\
 & Cierto
\end{array}
$$

Paso Inductivo

$$
\begin{array}{ll}
 & mapUrna\ f\ (U\ x\ u)\ =\ mapUrna'\ f\ (U\ x\ u) \\
\equiv & \{\ 2)mapUrna,\ 2)mapUrna'\ \} \\
 & U\ (f\ x)\ (mapUrna\ f\ u)\ =\ pl\ (\lambda\, x\ u\ \rightarrow\ U\ (f\ x)\ u)\ V\ (U\ x\ u) \\
\equiv & \{\ \text{h.i., } 2)pl\ \} \\
 & U\ (f\ x)\ (mapUrna'\ f\ u)\ =\ (\lambda\, x\ u\ \rightarrow\ U(f\ x)\ u)\ x\ (pl\ ...\ V\ u) \\
\equiv & \{\ \text{aplicación, y } mapUrna'\ \} \\
 & U\ (f\ x)\ (mapUrna'\ f\ u)\ =\ U(f\ x)\ (mapUrna'\ f\ u) \\
\equiv & \\
 & Cierto
\end{array}
$$

(E).–

$$sacoAUrna\ =\ foldr\ insertaEnUrna\ V$$

$$
\begin{array}{l}
insertaEnUrna\ (x,1)\qquad u\ =\ U\ x\ u \\
insertaEnUrna\ (x,n+1)\ u\ =\ U\ x\ (insertaEnUrna\ n\ u)
\end{array}
$$

$$cardinalSaco\ =\ foldr\ (\lambda\,(\_,n)\ p\ \rightarrow\ p+n)\ 0$$

$$urnaASaco\ =\ pl\ (\lambda\, x\ s\ \rightarrow\ insertaEnSaco\ x\ s)\ [\,]$$

$$
\begin{array}{ll}
insertaEnSaco\ x\ [\,] & =\ [(x,1)] \\
insertaEnSaco\ x\ ((y,n):s) & \\
\quad |\ x == y & =\ (x,n+1):s \\
\quad |\ otherwise & =\ (y,n)\ :\ insertaEnSaco\ x\ s
\end{array}
$$

$$cardinalUrna\ =\ pl\ (\lambda\, \_\ n\ \rightarrow\ n+1)\ 0$$

(F).– Es el test de pertenencia de un objeto a una urna.
(G).– $asco\ u\ u'$ comprueba el test de contenido $u'\ \subseteq\ u$.
(H).– Calcula, en una sola pasada, el último objeto de una urna, eliminando a su vez las repeticiones de éste.

Solución al Ejercicio 18.18 (pág. 521).– Se observa que, salvo el primero, la lista es suma de las listas:

$$zipWith\ (+)\ [1,\ 2,\ 4,\ 7,\ 11,\ 16,\ 22,\ 29,\ 37,\ \ldots]\ [1,\ 2,\ 3,\ 4,\ \ldots]$$

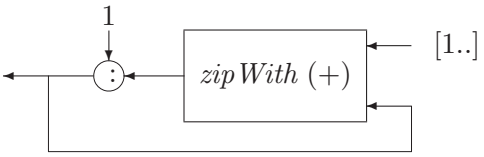

Figura 19.1: Red del Ejercicio 18.18..

$\Longrightarrow$
$[2,\ 4,\ 7,\ 11,\ \ldots]$

Luego la ecuación resultante es

$$sol\ =\ 1\ :\ zipWith\ (+)\ [1..]\ sol$$

y la red será la de la Figura 19.1.

Solución al Ejercicio 18.19 (pág. 522).– **(A)**.– Ejemplos pueden ser

$$eje1\ =\ \lambda\, x\ \rightarrow\ (\lambda\, y\ \rightarrow\ x)$$
$$eje2\ x\ =\ \lambda\, y\ \rightarrow\ y$$

(B).– Asociamos tipos arbitrarios a argumentos y resultado: $x\ ::\ a$, $y\ ::\ c$, $y\ x\ x\ ::\ b$, de donde $f_1\ ::\ a\ \rightarrow\ c\ \rightarrow\ b$. Pero de ser $y\ x\ x\ ::\ b$, por la regla de la aplicación y por igualdad de tipos tenemos $y\ ::\ a\ \rightarrow\ a\ \rightarrow\ b\ (\equiv\ c)$, de donde

$$f_1\ ::\ a\ \rightarrow\ (a\ \rightarrow\ a\ \rightarrow\ b)\ \rightarrow\ b$$

Ahora bien, si llamamos

$$T(a,b)\ \doteq\ (a\ \rightarrow\ a\ \rightarrow\ b)\ \rightarrow\ b$$

entonces $f_1\ ::\ a\ \rightarrow\ T(a,b)$, y ya que la composición tiene por tipo

$$(.)\ ::\ (v\ \rightarrow\ w)\ \rightarrow\ (u\ \rightarrow\ v)\ \rightarrow\ (u\ \rightarrow\ w)$$

debería tenerse, para la composición $f_1\,.\,f_1\ ::\ u\ \rightarrow\ w$, y podemos asociar los tipos $u\ \equiv\ a,\ v\ \equiv\ T(a,b),\ w\ \equiv\ T(v,c)$ de donde el tipo de f_2 es

$$\begin{array}{l} a\ \rightarrow\ T(\ T(a,b)\ ,\ c) \\ \equiv \\ a\ \rightarrow\ (T(a,b)\ \rightarrow\ T(a,b)\ \rightarrow\ c)\ \rightarrow\ c \\ \equiv \\ a\ \rightarrow\ (((a\ \rightarrow\ a\ \rightarrow\ b)\ \rightarrow\ b)\ \rightarrow\ ((a\ \rightarrow\ a\ \rightarrow\ b)\ \rightarrow\ b)\ \rightarrow\ c)\ \rightarrow c \end{array}$$

Solución al Ejercicio 18.20 (pág. 522).– **(A)**.–

```
paraTodo  xs p  =  and [ p x | x ← xs ]
existeAlgún xs p  =  or [ p x | x ← xs ]
```

(B).–

```
asociativa g op  =  and [ x `op` (y `op` z) == (x `op` y) `op` z |
                          x ← g, y ← g, z ← g ]

conmutativa g op  =  and [ x `op` y == y `op` x | x ← g, y ← g ]

neutroDcha e gs op  =  paraTodo gs (λ x → x `op` e == x)

neutroIzda e gs op  =  paraTodo gs (λ x → e `op` x == x)

neutro e gs op  =  neutroDcha e gs op && neutroIzda e gs op

neutros gs op  =  head [ neutro e gs op | e ← gs ]
– – tomamos la cabeza, ya que si existe neutro, es único

inverso x x' gs op  =  case neutros gs op of
                         []    → False
                         e : _ → x `op` x' == e && x' `op` x == e
grupo gs op = asociativa gs op && todosInvertibles
  where
    todosInvertibles  =  paraTodo gs p
    p x               =  [ x' | x' ← gs, inverso x x' gs op ] ≠ []
```

Solución al Ejercicio 18.21 (pág. 523).– **(A)**.–

```
separar []          = ([], [])
separar [y]         = ([y], [])
separar (x : y : ys) = (x : u, y : v) where (u, v) = separar ys

reunir ([], ys)         = ys
reunir (xs, [])         = xs
reunir (x : xs, y : ys) = x : y : reunir (xs, ys)
```

(B).–

```
maxmin []       = []
maxmin (x : xs) = reunir(su, sv)
  where
    (u, v) = separar (x : xs)
    su     = sustituye (máximo u) u
    sv     = sustituye (mínimo v) v
```

```
máximo [x]          = x
máximo (x : y : ys) = if x > y then
                        máximo (x : ys)
                      else
                        máximo (y : ys)
mínimo [x]          = x
mínimo (x : y : ys) = if x < y then
                        mínimo (x : ys)
                      else
                        mínimo (y : ys)
sustituye x [_]     = [x]
sustituye x (_ : ys) = x : sustituye x ys
```

(C).–

```
mkárbol [x]       = H x
mkárbol (x : xs)  = N (mkárbol u) (mkárbol v)
  where
    (u, v) = separar (x : xs)
```

(D).– Si numeramos de 0 en adelante:

```
elemento (H x)  _ = x
elemento (N i d) n = elemento r m
  where
    m = n `div` 2
    r = if (even n) then i else d
```

Solución al Ejercicio 18.22 (pág. 524).– La primera repite la lista $x +\!\!+ y$ tantas veces como elementos aparezcan en la lista x; si x es vacía devuelve la lista vacía:

```
MAIN> añade1 [ "blas", "lis" ] ["juan", "paco"]
["blas", "lis", "juan", "paco", "blas", "lis", "juan", "paco"]
```

La segunda añade x por la cabeza a la lista y si x es una lista no vacía; en otro caso devuelve la lista vacía:

```
MAIN> añade2 "blas" ["maria", "manolo"]
["blas", "maria", "manolo"]
```

La tercera devuelve $x +\!\!+ y$ si x es una lista no vacía; en otro caso devuelve la lista vacía:

```
MAIN> añade3 ["blas"] ["maria", "manolo"]
["blas", "maria", "manolo"]
```

Los tipos de las funciones son:

$$
\begin{aligned}
&añade1 :: [[a]] \to [[a]] \to [[a]]\\
&añade2 :: [a] \to [[a]] \to [[a]]\\
&añade3 :: [[a]] \to [[a]] \to [[a]]
\end{aligned}
$$

Solución al Ejercicio 18.23 (pág. 524).– **(A)**.–

$$
\begin{aligned}
&unión\ [\,]\ \ \ \ \ ys = ys\\
&unión\ (x : xs)\ ys = \mathbf{if}\ elem\ x\ ys\ \mathbf{then}\\
&\qquad\qquad\qquad\quad unión\ xs\ ys\\
&\qquad\qquad\qquad\ \mathbf{else}\\
&\qquad\qquad\qquad\quad x : unión\ xs\ ys
\end{aligned}
$$

(B).–

$$
\begin{aligned}
&variables\ (Y\ x\ y) = unión\ (variables\ x)\ (variables\ y)\\
&variables\ (O\ x\ y) = unión\ (variables\ x)\ (variables\ y)\\
&variables\ (AF\ x) = [x]\\
&variables\ (NG\ x) = [x]
\end{aligned}
$$

(C).–

$$
\begin{aligned}
&evalúa\ (Y\ x\ y)\ e = (evalúa\ x\ e)\ \&\&\ (evalúa\ y\ e)\\
&evalúa\ (O\ x\ y)\ e = (evalúa\ x\ e)\ ||\ (evalúa\ y\ e)\\
&evalúa\ (AF\ x)\ e = valor\ x\ e\\
&evalúa\ (NG\ x)\ e = not\ (valor\ x\ e)
\end{aligned}
$$

$$
\begin{aligned}
&valor\ x\ [\,]\ \ \ \ \ = False\\
&valor\ x\ (y : ys) = x == y\ ||\ valor\ x\ ys
\end{aligned}
$$

(D).–

$$
\begin{aligned}
&partes = foldr\ (\lambda\, x\ s \to map\ (x :)\ s +\!\!+ s)\ [[\,]]\\
&tautología\ e = (and\ .\ (map\ (evalúa\ e))\ .\ partes\ .\ variables)\ e
\end{aligned}
$$

Solución al Ejercicio 18.24 (pág. 524).– **(A)**.– Si asignamos tipos arbitrarios a argumentos y resultado a partir de la segunda ecuación

$$f :: a,\ h :: b,\ z :: c,\ (x : xs) :: [d],\ f(h\ x)\ (red\ f\ h\ z\ xs) :: e$$

entonces por la primera ecuación $h\ z :: e$, de donde $h :: c \to e$, y por la segunda $f :: e \to e \to e$. Y finalmente:

$$red :: (e \to e \to e) \to (c \to e) \to c \to [c] \to e$$

(B).– Procedemos por inducción sobre la lista xs.
—*Caso Base*:

$$
\begin{array}{ll}
 & 1 + lon\ [\,] = red\ (+)\ (\lambda\, x \rightarrow 1)\ 0\ [\,] \\
\equiv & \{\, 1)lon,\ 1)red \,\} \\
 & 1 + 0 = (\lambda\, x \rightarrow 1)\ 0 \\
\equiv & \{\, \beta\text{–regla} \,\} \\
 & Cierto
\end{array}
$$

—*Paso Inductivo*:

$$
\begin{array}{ll}
 & 1 + lon\ (x : xs) = red\ (+)\ (\lambda\, x \rightarrow 1)\ 0\ (x : xs) \\
\equiv & \{\, 2)lon,\ 2)red \,\} \\
 & 1 + 1 + lon\ xs = (+)\ ((\lambda\, x \rightarrow 1)\ x)\ (red\ (+)\ (\lambda\, x \rightarrow 1)\ 0\ xs) \\
\equiv & \{\, \beta\text{–regla} \,\} \\
 & 1 + 1 + lon\ xs = (+)\ 1\ (red\ (+)\ (\lambda\, x \rightarrow 1)\ 0\ xs) \\
\Leftarrow & \{\, is \,\} \\
 & 1 + lon\ xs = red\ (+)\ (\lambda\, x \rightarrow 1)\ 0\ xs \\
\equiv & \{\, \text{hipótesis de inducción} \,\} \\
 & Cierto
\end{array}
$$

(C).– El esquema de reducción puede expresarse vía $foldr$, ya que es posible probar fácilmente

$$(*) \qquad red\ f\ h\ z = foldr\ (f.h)\ (h\ z)$$

Para verlo es suficiente observar que

$$
\begin{array}{ll}
 & red\ f\ h\ z\ [x_1, x_2] \\
\equiv & \\
 & h\ x_1\ `f`\ (h\ x_2\ `f`\ h\ z) \\
\equiv & \\
 & x_1\ `f.h`\ (x_2\ `f.h`\ h\ z) \\
\equiv & \\
 & foldr\ (f.h)\ (h\ z)\ [x_1, x_2]
\end{array}
$$

(D).– Ya que

$$or = foldr\ (\|)\ True \qquad sum = foldr\ (+)\ 0$$

tomando en **(*)** $h \equiv id$, tendremos

$$or = red\ (\|)\ id\ True \qquad sum = red\ (+)\ id\ 0$$

Solución al Ejercicio 18.25 (pág. 525).–

$$
\begin{array}{l}
sol = iterate\ espiral\ (1, 0) \\
espiral\ (x, y) \\
\quad | \ x > 0 = (0, -x - 1) \\
\quad | \ x < 0 = (0, -x + 1) \\
\quad | \ y > 0 = (y + 1, 0) \\
\quad | \ y < 0 = (-y - 1, 0)
\end{array}
$$

Solución al Ejercicio 18.26 (pág. 525).– Asignamos tipos a argumentos y resultado

$$x :: a \qquad y :: b\ z :: c \qquad (x\ z)(y\ z)\ ::\ u$$

Entonces tendremos

$$\begin{array}{ll} & (x\ z)(y\ z)\ ::\ u \\ \Rightarrow & \{\ \text{regla de la aplicación}\ \} \\ & x\ z :: \alpha \to u,\ y\ z\ :\ \alpha \\ \Rightarrow & \{\ \text{regla de la aplicación}\ \} \\ & x\ ::\ c \to \alpha \to u,\ z\ ::\ c,\ y\ ::\ c \to \alpha \end{array}$$

y de aquí,

$$s\ ::\ (c \to \alpha \to u\ ) \to (c \to \alpha) \to c \to u$$

Una función con tipo $(a \to b) \to (a \to c) \to a \to (b, c)$ puede ser

$$ejemplo\ f\ g\ x\ =\ (f\ x,\ g\ x)$$

Solución al Ejercicio 18.27 (pág. 525).– **(A)**.–

$$\begin{array}{lll} numVar\ (P\ c) & = 1 & --\ 1) \\ numVar\ (N\ e) & = numVar\ e & --\ 2) \\ numVar\ (F\ e\ e') & = numVar\ e\ +\ numVar\ e' & --\ 3) \end{array}$$

(B).–

$$numVar' \ =\ plexpb\ (+)\ id\ (\lambda\_ \to 1)$$

(C).–

$$\begin{array}{ll} & \forall\ ex\ \centerdot\ ex\ ::\ ExpB\ \centerdot\ numVar\ ex\ =\ numVar'\ ex \\ \equiv & \{\ \text{principio de inducción}\ \} \\ & \quad --\ \textit{Caso Base} \\ & numVar\ (P\ c)\ =\ numVar'\ (P\ c) \\ \wedge & \\ & numVar\ e\ =\ numVar'\ e\ \wedge\ numVar\ e'\ =\ numVar'\ e' \\ & \quad \Rightarrow\ --\ \textit{Paso Inductivo} \\ & numVar\ (N\ e)\ =\ numVar'\ (N\ e)\ \wedge\ numVar\ (F\ e\ e')\ =\ numVar'\ (F\ e\ e') \end{array}$$

(D).– *Caso Base*:

$$\begin{array}{ll} & numVar\ (P\ c)\ =\ numVar'\ (P\ c) \\ \equiv & \{\ 1)numVar, \text{ def. } numVar'\ \} \\ & 1\ =\ plexpb\ (+)\ id\ (\lambda\_ \to 1)\ (P\ c) \\ \equiv & \{\ 1)plexpb\ \} \\ & 1\ =\ (\lambda\_ \to 1)\ c \end{array}$$

$\equiv$ { β regla }
$Cierto$

—*Paso Inductivo* I:

$numVar'\ (N\ e)$
$\equiv$ { definición $numVar'$ }
$plexpb\ (+)\ id\ (\lambda\_ \rightarrow 1)\ (N\ e)$
$\equiv$ { 2)$plexpb$ }
$id\ (plexpb\ (+)\ id\ (\lambda\_ \rightarrow 1)\ e)$
$\equiv$ { definición id }
$plexpb\ (+)\ id\ (\lambda\_ \rightarrow 1)\ e$
$\equiv$ { definición $numVar'$ }
$numVar'\ e$
$\equiv$ { hipótesis de inducción }
$numVar\ e$
$\equiv$ { 2)$numVar$ }
$numVar\ (N\ e)$

—*Paso Inductivo* II:

$numVar'\ (F\ e\ e')$
$\equiv$ { definición $numVar'$ }
$plexpb\ (+)\ id\ (\lambda\_ \rightarrow 1)\ (F\ e\ e')$
$\equiv$ { 3)$plexpb$ }
$(+)\ (plexpb\ (+)\ id\ (\lambda\_ \rightarrow 1)\ e)\ (plexpb\ (+)\ id\ (\lambda\_ \rightarrow 1)\ e')$
$\equiv$ { def. $numVar'$ }
$numVar'\ e\ +\ numVar'\ e'$
$\equiv$ { hipótesis de inducción }
$numVar\ e\ +\ numVar\ e'$
$\equiv$ { 3)$numVar$ }
$numVar\ (F\ e\ e')$

Solución al Ejercicio 18.28 (pág. 526).– **(A)**.–

$$solA\ =\ (1,2,3)\ :\ map\ (\lambda\ (x,y,z)\ \rightarrow\ (x+1, 2*x, x+y))\ solA$$

(B).– $solB\ =\ concat\ (\ map\ (\lambda\ (x,y,z)\ \rightarrow\ [x,y,z])\ solA)$

Solución al Ejercicio 18.29 (pág. 526).– Ver el problema de la sopa de letras en Capítulo 13 .

Solución al Ejercicio 18.30 (pág. 527).– **(A)**.– y **(B)**.– Escribiremos en primer lugar la función $mapMat$ del apartado (B), y a partir de ella las del apartado (A):

$$mapMat\ ::\ (Escalar\ \rightarrow\ Escalar)\ \rightarrow\ Matriz\ \rightarrow\ Matriz$$
$$mapMat\ f\ (MkMat\ xss)\ =\ MkMat\ (map\ (map\ f)\ xss)$$

```
negarMat :: Matriz → Matriz
negarMat = mapMat (*(-1.0))

porMat :: Escalar → Matriz → Matriz
porMat k = mapMat (*k)

esMat :: Matriz → Bool
esMat (MkMat [])  = False
esMat (MkMat xss) = todosIguales (map length xss)
  where
    todosIguales []     = True
    todosIguales (l:ls) = and [ y== l | y ← ls]
```

(C).–

```
porMixto :: Matriz → Vector → Vector
porMixto m@(MkMat fss) v@(MkVec es)
  | not (esMat m)   = error "Matriz no válida"
  | cols m ≠ dim v  = error "Matriz y vector no compatibles"
  | otherwise       = MkVec (
                        map (λfs → sum (zipWith (*) fs es)) fss)
  where
    cols (MkMat (f1:fss)) = length f1
    dim (MkVec es)        = length es


sumMat :: Matriz → Matriz → Matriz
sumMat m1@(MkMat xss) m2@(MkMat yss)
  | not (esMat m1 && esMat m2) = error "Matriz no válida"
  | dim m1 ≠ dim m2            = error "Matrices incompatibles"
  | otherwise                  = (MkMat . zipWith (zipWith (+)))
                                 xss yss
  where
    dim (MkMat fss) = (length fss, length (head fss))
```

(D).–

```
foldMat :: (Fila → a → a) → a → Matriz → a
foldMat f e (MkMat fss) = foldr f e fss

mayorMod :: Matriz → Fila
mayorMod = foldMat f []
  where
    f fs mayorFs = if módulo fs > módulo mayorFs then
                     fs
                   else
                     mayorFs
    módulo       = sum . map (^2)
```

19.3. AÑO 1998

Solución al Ejercicio 18.31 (pág. 528).– **(A)**.– Obsérvese que tanto f como g son funciones de dos y un argumento respectivamente; si les asignamos a dichas funciones tipos arbitrarios, $f :: a \to b \to c$, $g :: d \to e$, tendremos, por la primera ecuación y por la segunda, que los tipos de $f\ x\ q$ y $g\ x$ deben coincidir, por tanto $c \equiv e$ y $a \equiv d$. Pero también q debe ser del mismo tipo que el tipo base de la lista respuesta; luego finalmente, $a \equiv b$, $e \equiv b$, y tendremos

$$recons :: (a \to b \to b) \to (a \to b) \to [a] \to [b]$$

(B).–

$$\begin{array}{l}
\forall f, g, xs \bullet f :: a \to b \to b,\ g :: a \to b,\ xs :: [a], xs \neq [\,] \bullet \\
\quad length\ (recons\ f\ g\ xs) = length\ xs \\
\equiv \quad \{\text{ principio de inducción (el conjunto de listas no vacías es inductivo) }\} \\
\quad \textit{-- Caso Base} \\
\quad \forall f, g, x \bullet f :: a \to b \to b,\ g :: a \to b,\ x :: a \bullet \\
\quad\quad length\ (recons\ f\ g\ [x]) = length\ [x] \\
\wedge \\
\quad \forall f, g \bullet f :: a \to b \to b,\ g :: a \to b \bullet \\
\quad \forall xs, x \bullet x :: a,\ xs :: [a] \bullet \\
\quad\quad length\ (recons\ f\ g\ xs) = length\ xs \\
\quad\quad \Rightarrow \quad \textit{-- Paso Inductivo} \\
\quad\quad length\ (recons\ f\ g\ (x : xs)) = length\ (x : xs)
\end{array}$$

(C).– *Caso Base*:

$$\begin{array}{l}
length\ (recons\ f\ g\ [x]) = length\ [x] \\
\equiv \quad \{\, 1)recons,\ 2)length \,\} \\
length\ [g\ x] = 1 + length\ [\,] \\
\equiv \quad \{\, 2)length,\ 1)length \,\} \\
1 + length\ [\,] = 1 \\
\equiv \quad \{\, 2)length \,\} \\
Cierto
\end{array}$$

Paso Inductivo:

$$\begin{array}{l}
length\ (recons\ f\ g\ (x : xs)) \\
\equiv \quad \{\, 2)recons \,\} \\
length\ (f\ x\ q\ :\ qs\ \textbf{where}\ qs@(q : _) = recons\ f\ g\ xs) \\
\equiv \quad \{\, 2)length \,\} \\
1 + length\ (recons\ f\ g\ xs) \\
\equiv \quad \{\text{ hipótesis de inducción }\} \\
1 + length\ xs \\
\equiv \quad \{\, 2)length \,\} \\
length\ (x : xs)
\end{array}$$

(D).–

```
suma = head . recons (+) id
máximo = head . recons max id
reemPorÚltimo = recons (λ x y → y) id
numera = recons (λ x (n, _) → (n + 1, x) ) (λ y → (1, y))
```

(E).–

```
ruffini x0 xs = (v, rs)
  where v : rs = recons (λ x y → x + y ∗ x0) id xs
```

(F).–

```
segundoMenor = (λ (_, v) → v) . head .
      recons (λ x (m, m2) → if x≤ m then
                               (x, m)
                            else
                               if x≤ m2 then
                                  (m, x)
                               else
                                  (m, m2) )
      (λ y → (y, y) )
```

Solución al Ejercicio 18.32 (pág. 529).– **(A).–**

```
primo 1 = False
primo n = and [ n `mod` d ≠ 0 | d ← [2..n − 1] ]
divisoresPrimos :: Int → [Int]
  −− divisoresPrimos n ⟹ la lista de divisores primos de n
divisoresPrimos n = [ p | p ← [2..n], n `mod` p == 0, primo p ]
```

(B).–

```
gradoDe n p
  | n `mod` p ≠ 0 = 0
  | otherwise     = 1 + gradoDe (n `div` p) p
factorizacion n = [ (p, gradoDe n p) | p ← divisoresPrimos n]
```

(C).–

```
mcd a b = foldr (λ (p, e) v → p ↑ e ∗ v) 1
  [ (p, min e e′) | (p, e) ← factorizacion a,
                    (p′, e′) ← factorizacion b,
                    p′ == p]
```

De la misma manera también se podría calcular el mínimo común múltiplo:

```
mcm a b = foldr (λ(p, e) v → p ↑ e * v) 1 (cync f f')
  where
    f = factorizacion a
    f' = factorizacion b
-- comunes con mayor exponente y no comunes

cync []          ws'          = ws'
cync ws          []           = ws
cync ws@((p, e) : ps) ws'@((p', e') : ps')
  | p == p'                   = (p, max e e') : cync ps ps'
  | p < p'                    = (p, e) : cync ps ws'
  | otherwise                 = (p', e') : cync ws ps'
```

Solución al Ejercicio 18.33 (pág. 530).– Hay que demostrar, por inducción sobre la lista xs

$$\forall\, xs \;.\; xs :: [a] \;.\; long\ (map\ f\ xs) = long\ xs$$

—*Caso Base*:

$$\begin{array}{ll} & long\ (map\ f\ [\,]) = long\ [\,] \\ \equiv & \{\ 1)map,\ 1)long\ \} \\ & long\ [\,] = 0 \\ \equiv & \{\ 1)long\ \} \\ & Cierto \end{array}$$

—*Paso Inductivo*:

$$\begin{array}{ll} & long\ (map\ f\ (x : xs)) = long\ (x : xs) \\ \equiv & \{\ 2)map,\ 2)long\ \} \\ & long\ (f\ x\ :\ map\ f\ xs) = 1 + long\ xs \\ \equiv & \{\ 2)long\ \} \\ & 1 + long(map\ f\ xs) = 1 + long\ xs \\ \equiv & \{\ \text{hipótesis de inducción}\ \} \\ & Cierto \end{array}$$

Solución al Ejercicio 18.34 (pág. 530).– **(A)**.–

```
solA = san where
  san      = 1 : sanm1
  sanm1    = 1 : sanm2
  sanm2    = 1 : sanm3
  sanm3    = zipWith4 f [0..] san sanm1 sanm2
  f n x y z = n * x + 2 * (n + 1) * y + 3 * (n + 2) * z
```

donde recordamos que $zipWith4$ es la función del módulo LIST definida en la forma

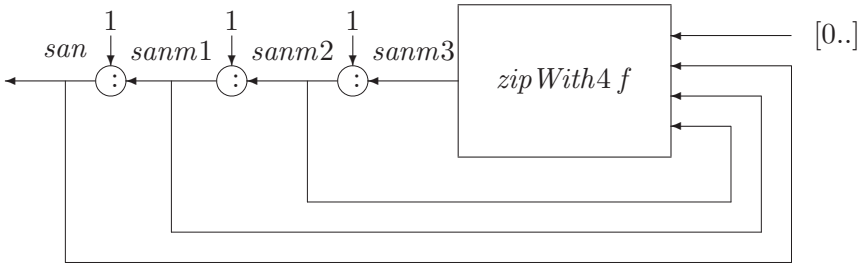

Figura 19.2: Red correspondiente al Ejercicio 18.34..

$$zipWith4\ f\ (x : xs)\ (y : ys)\ (z : zs)\ (u : us)\ =\ f\ x\ y\ z\ u\ :\ zipWith4\ f\ xs\ ys\ zs\ us$$

(B).–

$$\begin{array}{l} menor\ =\ menor'\ solA\ 2 \\ menor'\ (x : xs'@(y : z : xs))\ n \mid z + y - x\ >\ 100000000\ =\ n \\ \qquad\qquad\qquad\qquad\qquad\quad \mid otherwise\qquad\qquad\ =\ menor'\ xs'\ (n+1) \end{array}$$

Solución al Ejercicio 18.35 (pág. 530).– **(A)**.–

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

(B).–

$$\begin{array}{l} ej1\ f\ g\ h\ x\ =\ g\ (h\ x) \\ ej2\ f\ g\ h\ x\ =\ f\ (h\ x)\ x \end{array}$$

Solución al Ejercicio 18.36 (pág. 530).– **(A)**.– $f \equiv \lambda\, r\ e\ \to\ r +\!\!+ [e+1]$, $z \equiv [4]$.
(B).– Siendo f la función anterior, entonces:

$$\begin{array}{l} \forall\ u :: [a]\ \centerdot\ map\ (+1)\ (inv\ u)\ =\ foldr\ f\ [\,]\ u \\ \equiv\!\!\equiv \\ \quad -- \textit{Caso Base} \\ \quad map\ (+1)\ (inv\ [\,])\ =\ foldr\ f\ [\,]\ [\,] \\ \wedge \\ \quad -- \textit{Paso Inductivo} \\ \quad \forall\ u :: [a],\ x :: a\ \centerdot \\ \qquad map\ (+1)\ (inv\ u)\ =\ foldr\ f\ [\,]\ u \\ \quad\ \Rightarrow \\ \qquad map\ (+1)\ (inv\ (x : u))\ =\ foldr\ f\ [\,]\ (x : u) \end{array}$$

(C).– *Caso Base*:

$$
\begin{array}{ll}
 & map\ (+1)\ (inv\ [\,])\ =\ foldr\ f\ [\,]\ [\,] \\
\equiv & \{\ 1)inv,\ 1)foldr\ \} \\
 & map\ (+1)\ [\,]\ =\ [\,] \\
\equiv & \{\ 1)map\ \} \\
 & [\,]\ =\ [\,] \\
\equiv & \\
 & Cierto
\end{array}
$$

—*Paso Inductivo*:

$$
\begin{array}{ll}
 & map\ (+1)\ (inv\ (x : u))\ =\ foldr\ f\ [\,]\ (x : u) \\
\equiv & \{\ 2)inv,\ 2)foldr\ \} \\
 & map\ (+1)\ (inv\ u +\!\!+ [x])\ =\ f\ x\ (foldr\ f\ [\,]\ u) \\
\equiv & \{\ \text{por la propiedad (*), y definición de } f\ \} \\
 & map\ (+1)\ (inv\ u) +\!\!+ map\ (+1)[x]\ =\ foldr\ f\ [\,]\ u +\!\!+ [x+1] \\
\equiv & \{\ 2)map\ \} \\
 & map\ (+1)\ (inv\ u) +\!\!+ (x+1) : map\ (+1)[\,]\ = foldr\ f\ [\,]\ u +\!\!+ [x+1] \\
\equiv & \{\ 1)map\ \} \\
 & map\ (+1)\ (inv\ u) +\!\!+ [x+1]\ =\ foldr\ f\ [\,]\ u +\!\!+ [x+1] \\
\Leftarrow & \{\ \text{is}\ \} \\
 & map\ (+1)\ (inv\ u)\ =\ foldr\ f\ [\,]\ u \\
\equiv & \{\ \text{hipótesis de inducción}\ \} \\
 & Cierto
\end{array}
$$

Solución al Ejercicio 18.37 (pág. 531).– **(A)**.–

$$
\begin{array}{l}
parDropWhile\ p\ xs@(x : y : ys) \\
\quad \mid p\ x\ y \qquad\qquad = parDropWhile\ p\ (y : ys) \\
\quad \mid otherwise \qquad = xs \\
parDropWhile\ p\ \_\ = [\,]
\end{array}
$$

(B).– $parDropUntil\ p\ xs\ =\ parDropWhile\ (\lambda\ x\ y\ \rightarrow\ not\ (p\ x\ y))\ xs$

Solución al Ejercicio 18.38 (pág. 531).– La red aparece en la Figura 19.3.

$$red\ =\ [1] : map\ (concat.(map\ (\lambda\ x \rightarrow\ [x-1, x+1])))\ red$$

Solución al Ejercicio 18.39 (pág. 531).– **(A)**.–

$$
\begin{array}{l}
mapPoli\ ::\ (Monomio\ \rightarrow\ Monomio)\ \rightarrow\ Polinomio\ \rightarrow\ Polinomio \\
mapPoli\ f\ PoliNulo \qquad = PoliNulo \\
mapPoli\ f\ (m\ :+:\ p)\ =\ f\ m\ :+:\ mapPoli\ f\ p
\end{array}
$$

(B).–

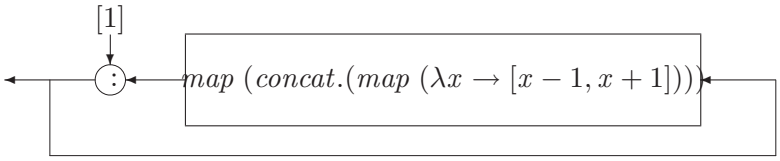

Figura 19.3: Red correspondiente al Ejercicio 18.38..

```
evalPoli :: Float → Polinomio → Float
evalPoli _ PoliNulo          = 0.0
evalPoli x ((c, g) :+: p) = c ∗ x^g + evalPoli x p
```

(C).–

```
evalPoli' :: Float → Polinomio → Float
evalPoli' x = pliegaPoli (λ (c, g) e → c ∗ x ∧ g + e) 0.0
```

(D).–

```
infixl 6 <+>
(<+>) :: Polinomio → Polinomio → Polinomio

PoliNulo                  <+> p2                    = p2
p1                        <+> PoliNulo              = p1
p1@((c1, g1) :+: pr1) <+> p2@((c2, g2) :+: pr2)
  | g1 > g2  = (c1, g1) :+: (pr1 <+> p2)
  | g2 > g1  = (c2, g2) :+: (p1 <+> pr2)
  | otherwise = (c1 + c2, g1) :+: (pr1 <+> pr2)
```

(E).–

```
infixl 7 <∗>
(<∗>) :: Polinomio → Polinomio → Polinomio
(<∗>) p1 = pliegaPoli (λ m p → m `por` p1 <+> p) PoliNulo

-- producto de monomio por polinomio
por :: Monomio → Polinomio → Polinomio
_ `por` PoliNulo          = PoliNulo
m@(c, g) `por` ((c', g') :+: p') = (c ∗ c', g + g') :+: por m p'
```

Otra posibilidad, vía plegado:

```
por' (c, g) = pliegaPoli (λ (c', g') p → (c ∗ c', g + g') :+: p) PoliNulo
```

Trata ahora de averiguar qué hacen las siguientes funciones:

```
evalPoli'' x = pliegaPoli (λ(c,g) e → x * evalResto + c) 0.0

spanPoli :: (Monomio → Bool) → Polinomio → (Polinomio, Polinomio)
spanPoli f PoliNulo = (PoliNulo, PoliNulo)
spanPoli f (m :+: p)
  | f m       = (m :+: izq, der)
  | otherwise = (izq, m :+: der)
  where
    (izq, der) = spanPoli f p


agrupa :: Polinomio → Polinomio
agrupa PoliNulo = PoliNulo
agrupa p@((_,g) :+: _) = (sumar iguales, g) :+: agrupa distintos
  where
    (iguales, distintos) = spanPoli (λ(_,g') → g == g') p
    sumar = pliegaPoli (λ(c,_) sumaResto → c + sumaResto) 0.0


esCero :: Polinomio → Bool
esCero = pliegaPoli (λ(c,g) b → c == 0.0 && b) True
```

Solución al Ejercicio 18.40 (pág. 532).– Asociamos tipos a argumentos y al resultado

$$g :: t1,\ x :: t2,\ (:)\ x\ (f\ g\ (g\ x)) \ ::\ t3$$

de tal forma que $(:) :: t4 \to [t4] \to [t4]$, $f :: t1 \to t2 \to t3$. Y entonces obtenemos, sucesivamente:

$$\begin{array}{l}
\exists\, t5 \mid f\ g\ (g\ x) :: t5 \wedge (:)\ x :: t5 \to t3 \\
\exists\, t6 \mid x :: t6 \wedge (:) :: t6 \to t5 \to t3 \\
\exists\, t7 \mid (g\ x) :: t7 \wedge f\ g :: t7 \to t5 \\
\exists\, t8 \mid x :: t8 \wedge g :: t8 \to t7 \\
\exists\, t9 \mid g :: t9 \wedge f :: t9 \to t7 \to t5
\end{array}$$

Y de las anteriores tendremos

$$\begin{array}{ll}
g :: t1 \wedge g :: t9 \wedge g :: t8 \to t7 & \Rightarrow t1 = t9 = t8 \to t7 \\
x :: t2 \wedge x :: t6 \wedge x :: t8 & \Rightarrow t2 = t6 = t8 \\
\left.\begin{array}{l} f :: t1 \to t2 \to t3 \\ f :: t9 \to t7 \to t5 \end{array}\right\} & \Rightarrow t1 = t9 \wedge t2 = t7 \wedge t3 = t5 \\
\left.\begin{array}{l} (:) :: t4 \to [t4] \to [t4] \\ (:) :: t6 \to t5 \to t3 \end{array}\right\} & \Rightarrow t4 = t6 \wedge t5 = t3 = [t4]
\end{array}$$

y de aquí $t8 \equiv t7 \equiv t4$, y finalmente

$$f :: (t4 \to t4) \to t4 \to [t4]$$

Solución al Ejercicio 18.41 (pág. 533).– **(A)**.– $[12,\ 13,\ 14,\ 15,\ 16,\ 14,\ 15,\ 16,\ 17,\ 18]$

(B).–

$$
\begin{array}{l}
f1\ h\ p\ [\,]\quad\ ys\ =\ [\,] \\
f1\ h\ p\ (x : xs)\ ys \\
\quad \mid p\ x\ \ \ \ \ \ \ \ \ \ \ \ =\ map\ (h\ x)\ ys +\!\!+ resto \\
\quad \mid otherwise\ \ \ \ \ \ =\ resto \\
\quad \textbf{where} \\
\qquad resto\ =\ f1\ h\ p\ xs\ ys
\end{array}
$$

$$
\begin{array}{l}
f2\ h\ p\ xs\ ys\ = \\
\quad concat\ (map\ (\lambda\, x\ \rightarrow\ \textbf{if}\ p\ x\ \textbf{then}\ map\ (h\ x)\ ys\ \textbf{else}\ [\,])\ xs)
\end{array}
$$

$$
f3\ h\ p\ xs\ ys\ =\ concat\ (map\ (\lambda\, x\ \rightarrow\ map\ (h\ x)\ ys)\ (filter\ p\ xs))
$$

Solución al Ejercicio 18.42 (pág. 533).– **(A)**.–

$$
\begin{array}{l}
\forall\ bs\ ::\ [t],\ as\ ::\ [t],\ xs\ ::\ [[t]]\ \centerdot \\
\qquad bs +\!\!+ foldl\ (+\!\!+)\ as\ xs\ =\ foldl\ (+\!\!+)\ (bs +\!\!+ as)\ xs \\
\equiv\ \{\ \text{inducción sobre } xs\ \} \\
\quad \textit{– – Caso Base}: \\
\forall\ bs\ ::\ [t],\ as\ ::\ [t]\ \centerdot \\
\qquad bs +\!\!+ foldl\ (+\!\!+)\ as\ [\,]\ =\ foldl\ (+\!\!+)\ (bs +\!\!+ as)\ [\,] \\
\wedge \\
\quad \textit{– – Paso Inductivo}: \\
\forall\ xs\ ::\ [[t]]\centerdot \\
\qquad (\ \forall\ bs\ ::\ [t],\ as\ ::\ [t]\centerdot \\
\qquad\qquad bs +\!\!+ foldl\ (+\!\!+)\ as\ xs\ =\ foldl\ (+\!\!+)\ (bs +\!\!+ as)\ xs\ ) \\
\quad \Rightarrow \\
\qquad (\forall\ bs :: [t],\ as\ ::\ [t],\ x :: [t]\ \centerdot \\
\qquad\qquad bs +\!\!+ foldl\ (+\!\!+)\ as\ (x : xs)\ =\ foldl\ (+\!\!+)\ (bs +\!\!+ as)\ (x : xs))
\end{array}
$$

(B).– Consideremos la función

$$
\begin{array}{lll}
foldl\ f\ z\ [\,]\ \ \ \ \ \ \ =\ z & & \text{– – (1)} \\
foldl\ f\ z\ (x : xs)\ =\ foldl\ f\ (f\ z\ x)\ xs & & \text{– – (2)}
\end{array}
$$

Aplicamos el esquema anterior:
—*Caso Base*:

$$
\begin{array}{l}
bs +\!\!+ foldl\ (+\!\!+)\ as\ [\,]\ =\ foldl\ (+\!\!+)\ (bs +\!\!+ as)\ [\,] \\
\equiv\ \{\ 1),\ 1)\ \} \\
bs +\!\!+ as\ =\ bs +\!\!+ as
\end{array}
$$

—*Paso Inductivo*:

$$
\begin{array}{l}
\quad bs +\!\!+ foldl\ (+\!\!+)\ as\ (x : xs) \ = \ foldl\ (+\!\!+)\ (bs +\!\!+ as)\ (x : xs) \\
\equiv \ \{\, 2), 2) \,\} \\
\quad bs +\!\!+ foldl\ (+\!\!+)\ ((+\!\!+)\ as\ x)\ xs \ = \ foldl\ (+\!\!+)\ ((+\!\!+)\ (bs +\!\!+ as)\ x)\ xs \\
\equiv \ \{\, \text{h.i., } +\!\!+ \text{ infija} \,\} \\
\quad foldl\ (+\!\!+)\ (bs +\!\!+ ((+\!\!+)\ as\ x))\ xs \ = \ foldl\ (+\!\!+)\ ((bs +\!\!+ as) +\!\!+ x)\ xs \\
\equiv \ \{\, +\!\!+ \text{ infija, asociatividad de } +\!\!+ \,\} \\
\quad foldl\ (+\!\!+)\ (bs +\!\!+ (as +\!\!+ x))\ xs \ = \ foldl\ (+\!\!+)\ (bs +\!\!+ (as +\!\!+ x))\ xs
\end{array}
$$

Solución al Ejercicio 18.43 (pág. 533).– **(A)**.–

```
red = r0
  where
    r0 = 0 : r1
    r1 = 1 : r2
    r2 = 2 : r3
    r3 = zipWith (+) (zipWith (+) r0 r1) r2
```

(B).–

```
red2 = tr red where tr (x : y : z : xs) = [z, y, x] : tr (y : z : xs)
```

Solución al Ejercicio 18.44 (pág. 533).– **(A)**.–

```
buscar :: Entorno -> NombreVar -> Valor
buscar [] y = error (y ++ " no definida")
buscar ((z, w) : xs) y
  |z== y      = w
  |otherwise  = buscar xs y
```

(B).–

```
asignar :: Entorno -> NombreVar -> Valor -> Entorno
asignar []           x y = [(x, y)]
asignar ((z, w) : zs) x y
  | x== z      = (z, y) : zs
  | otherwise  = (z, w) : asignar zs x y
```

(C).–

```
evalExpr :: Entorno -> Expr -> Valor
evalExpr xs (Const y) = y
evalExpr xs (Var y)   = buscar xs y
evalExpr xs (a :+ b)  = (evalExpr xs a) + (evalExpr xs b)
evalExpr xs (a :- b)  = (evalExpr xs a) - (evalExpr xs b)
evalExpr xs (a :* b)  = (evalExpr xs a) * (evalExpr xs b)
evalExpr xs (a :/ b)  = (evalExpr xs a) `div` (evalExpr xs b)
```

(D).– (y **(E).**–)

```
runProg    :: Programa → Entorno
runProg    =  runProg' []
runProg' e =  foldl runSent e

data Sentencia =  NombreVar := Expr
               | Inc NombreVar
               | For (NombreVar, Expr, Expr) [Sentencia]
                 deriving Show
prog2 = ["sum" := Const 0,
         For("i", Const 1, Const 5)
           [
             "sum" := Var "sum" :+ Var "i"
           ]
        ]
```

El siguiente bucle a su salida deja la variable al último valor, y además evalúa los límites una sola vez:

```
runSent :: Entorno → Sentencia → Entorno
runSent xs (y := sen)       = asignar xs y (evalExpr xs sen)
runSent xs (Inc y)          = asignar xs y ((buscar xs y) + 1)
runSent xs (For (v, i, f) ss) =
        foldl (λ e x → runProg' (asignar e v x) ss) xs [vi..vf]
  where
    vi = evalExpr xs i
    vf = evalExpr xs f
```

Solución al Ejercicio 18.45 (pág. 535).– **(A).**–

```
prefijo :: String → String → Bool
prefijo []     _      = True
prefijo _      []     = False
prefijo (a:as) (b:bs) = a== b && prefijo as bs
```

A partir de la anterior, la función *buscaPrefijosDesde* es fácil:

```
buscaPrefijosDesde _   _  []          = []
buscaPrefijosDesde pos as bs@(b:bs') =
     [ pos | prefijo as bs ] ++ buscaPrefijosDesde (pos + 1) as bs'
```

Observa que la primera lista de la línea anterior es vacía si en la posición *pos* no aparece tal prefijo. Aún así, seguimos buscando otra solución en el resto de la lista, pero a partir de la siguiente posición $pos + 1$.

(B).– Realmente es difícil intuir el algoritmo. Nos podemos ayudar con algún diálogo:

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

MAIN> $sorpresa\ miTabla$
$[[1],[2,2],[3,3,3],[4,4,4],[5,5,5],[6,6],[7]]$

MAIN> $miSopa$
["casacasa", "æapesaxx", "mxxaxxxx"]

MAIN> $sorpresa\ miSopa$
["c", "æa", "mas", "xpa", "xec", "æsa", "xas", "xxa", "xx", "x"]

A la vista del primero, parece que calcula las diagonales secundarias de la tabla. Para el segundo es fácil comprobar que también aparecen dichas diagonales:

$$\begin{array}{cccccccc} c & a & s & a & c & a & s & a \\ a & a & p & e & s & a & x & x \\ m & x & x & a & x & x & x & x \end{array}$$

19.4. AÑO 1999

Solución al Ejercicio 18.46 (pág. 536).– **(A)**.–

$$\begin{array}{l} eje1 \ = \ \lambda f\ x \ \rightarrow \ f\ x \\ eje2 \ = \ \lambda x\ f \ \rightarrow \ f\ x \end{array}$$

(B).– Asignando tipos a los argumentos y al resultado

$$g \ :: \ u \qquad x :: \ a \qquad dup\ g\ (g\ x) \ :: \ b$$

vemos que $dup \ :: \ u \ \rightarrow \ a \ \rightarrow \ b$. Pero en la expresión $dup\ g\ (g\ x)$ el segundo argumento $g\ x$ debe ser del mismo tipo que el de x; por tanto, u (el tipo de g) debe ser $a \ \rightarrow \ a$, de donde

$$dup \ :: \ (a \ \rightarrow \ a) \ \rightarrow \ a \ \rightarrow \ b$$

(C).– Si escribimos

$$dup2\ h \ = \ dup\ (\ dup\ h)$$

el tipo de h debe ser de la forma $a \ \rightarrow \ a$, y en ese caso $dup\ (dup\ h) \ :: \ a \ \rightarrow \ b$; por tanto

$$dup2 \ :: \ (a \ \rightarrow \ a) \ \rightarrow \ a \ \rightarrow \ b$$

Solución al Ejercicio 18.47 (pág. 537).– **(A)**.–

$$\begin{array}{l} \quad cur\ [a] \\ \equiv\equiv \\ \quad foldr\ inc\ [\,]\ [a] \\ \equiv\equiv \end{array}$$

$$
\begin{array}{ll}
& inc\ a\ [\,] \\
\equiv & \\
& a
\end{array}
$$

$$
\begin{array}{ll}
& cur\ [a, b] \\
\equiv & \\
& foldr\ inc\ [\,]\ (a : [b]) \\
\equiv & \\
& inc\ a\ (foldr\ inc\ [\,]\ [b]) \\
\equiv & \{\ \text{por lo anterior}\ \} \\
& inc\ a\ [b] \\
\equiv & \\
& [a + b, b]
\end{array}
\qquad
\begin{array}{ll}
& cur\ [a, b, c] \\
\equiv & \\
& foldr\ inc\ [\,]\ (a : [b, c]) \\
\equiv & \\
& inc\ a\ (foldr\ inc\ [\,]\ [b, c]) \\
\equiv & \{\ \text{por lo anterior}\ \} \\
& inc\ a\ [b + c, c] \\
\equiv & \\
& [a + b + c, b + c, c]
\end{array}
$$

(B).– Tomemos una lista no vacía $y : ys$, y calculemos

$$
\begin{array}{ll}
& head(inc\ x\ (y : ys)) \\
\equiv & \{\ inc\ \} \\
& head\ (x + y : (y : ys)) \\
\equiv & \{\ head\ \} \\
& x + y \\
\equiv & \{\ head\ \} \\
& x +\ head\ (y : ys)
\end{array}
$$

(C).– Para el caso base basta aplicar (A), ya que

$$head(cur\ [x])\ =\ head\ [x]\ =\ x\ =\ sum\ [x]$$

Para probar el paso inductivo, tendremos

$$
\begin{array}{ll}
& head(cur(x : xs)\ ) \\
\equiv & \{\ cur\ \} \\
& head(foldr\ inc\ [\,]\ (x : xs)) \\
\equiv & \{\ foldr\ \} \\
& head(inc\ x\ (foldr\ inc\ [\,]\ xs)) \\
\equiv & \{\ \text{(B)}\ \} \\
& x\ +\ head(foldr\ inc\ [\,]\ xs) \\
\equiv & \{\ \text{hipótesis de inducción}\ \} \\
& x\ +\ sum\ xs \\
\equiv & \{\ 2)sum\ \} \\
& sum\ (x : xs)
\end{array}
$$

Solución al Ejercicio 18.48 (pág. 537).– **(A)**.–

$$
\begin{array}{lll}
unidad\ 0 & (m+1) & = [\,] \\
unidad\ (n+1) & (m+1) & = (1.0 :\ replicate\ m\ 0.0)\ :\ map\ (0.0 :)\ (unidad\ n\ m) \\
unidad\ n & 0 & = replicate\ n\ [\,]
\end{array}
$$

```
nula n m = replicate n (replicate m 0.0)

diagonal :: Matriz → Fila
diagonal ((x : xs) : fs)      = x : diagonal (map tail fs)
diagonal ([] : _)             = []
diagonal []                   = []
```

(B).– La función *desconocida* computa la suma de matrices.

(C).–

```
pliegaM f g [x]      = g x
pliegaM f g (x : fs) = f x (pliegaM f g fs)
```

(D).–

```
módulo = sqrt . sum . map (↑ 2)

mayorMódulo = pliegaM máxima id
  where maxima f1 f2 = if módulo f1 > módulo f2 then f1 else f2

mapM f = pliegaM (λ x y → (map f x) : y) (λ u → [map f u])
```

Solución al Ejercicio 18.49 (pág. 538).– **(A).–**

```
sumaDígitos x y a = (s `div` 10, s `mod` 10) where s = x + y + a
```

(B).–

```
suma xs ys = if af == 0 then ds else af : ds
  where
    (af, ds) = foldr g (0, []) (zip xs ys)
    g (x, y) (a, ds) = (a', s : ds) where (a', s) = sumaDígitos x y a
```

(C).–

```
suma' :: [Int] → [Int] → [Int]
suma' xs ys = suma xs' ys' where (xs', ys') = completa xs ys
completa xs ys
    | dif > 0    = (xs, replicate dif 0 ++ ys)
    | otherwise = (replicate (−dif) 0 ++ xs, ys)
  where dif = length xs − length ys
```

(D).–

```
puzzleApartadoD =
        [ (a, b, c, d) | a ← ds,
                         b ← ds \\ [a],
                         c ← ds \\ [a, b],
                         d ← ds \\ [a, b, c],
                         suma [a, b, c, d] [d, a, b, c] == [c, d, a, b] ]
```

donde la función \\ calcula la diferencia de listas:

```
(\\) :: Eq a ⇒ [a] → [a] → [a]
[]     \\ _  = []
(x : xs) \\ ys
  | x `elem` ys  = xs \\ ys
  | otherwise    = x : (xs \\ ys)
```

La ejecución de la función anterior devuelve la lista vacía:

```
MAIN> puzzleApartadoD
[] :: [(Int, Int, Int, Int)]
```

por lo que el siguiente puzzle no tiene solución:

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

Observa que en cada línea aparece el mismo conjunto de letras pero permutadas.
(E).– La siguiente función genera algunos puzzles del tipo siguiente, donde los dígitos correspondientes a las letras se eligen de una lista ds:

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

y las líneas desconocidas son permutaciones de las letras de la primera fila:

```
generaPuzzles = [ ([a, b, c], ds, ds') | a ← ds,
                                         b ← ds \\ [a],
                                         c ← ds \\ [a, b],
                                         let pers = permuta [a, b, c],
                                         ds ← pers,
                                         ds' ← pers,
                                         suma [a, b, c] ds == ds' ]
```

Observa que lo que hacemos es *estudiar las permutaciones de los símbolos de la primera fila* con objeto de comprobar si con alguna permutación se obtiene un puzzle correcto. La función que calcula las permutaciones es ya conocida:

$$
\begin{array}{l}
permuta\ [x] \quad = [\,[x]\,] \\
permuta\ (x : xs) \ = \ concat\ (map\ (x+:)\ (permuta\ xs)) \\
x +: [\,] \quad = [[x]] \\
x +: xs@(y : ys) \ = \ (x : xs) \ : \ map\ (y :)\ (x +: ys)
\end{array}
$$

La ejecución de la función anterior produce esencialmente una solución:

$$
\begin{array}{l}
\text{MAIN}> \ generaPuzzles \\
[([4,5,9],[4,9,5],[9,5,4]),([4,9,5],[4,5,9],[9,5,4])]
\end{array}
$$

ya que la segunda lista es la misma solución (por ser la suma conmutativa). Si permitimos el dígito 0 aparecen otras soluciones nuevas

$$
\begin{array}{l}
\text{MAIN}> \ ds \\
[0,1,2,3,4,5,6,7,8,9]
\end{array}
$$

$$
\begin{array}{l}
\text{MAIN}> \ generaPuzzles \\
[([0,4,5],[4,0,5],[4,5,0]),([0,5,4],[4,5,0],[5,0,4]), \\
([4,0,5],[0,4,5],[4,5,0]),([4,5,0],[0,5,4],[5,0,4]), \\
([4,5,9],[4,9,5],[9,5,4]),([4,9,5],[4,5,9],[9,5,4])]
\end{array}
$$

Solución al Ejercicio 18.51 (pág. 539).– **(A)**.–

$$
\begin{array}{ll}
 & \forall\ xs\ ::\ [t],\ \bullet \\
 & \qquad z\ `f`\ foldl\ g\ y\ xs \ = \ foldl\ g\ (z\ `f`\ y)\ xs \\
\equiv & \{\ \text{inducción sobre } xs\ \} \\
 & \quad -- \textit{Caso Base}: \\
 & \qquad z\ `f`\ foldl\ g\ y\ [\,] \ = \ foldl\ g\ (z\ `f`\ y)\ [\,] \\
\wedge & \\
 & \quad -- \textit{Paso Inductivo}: \\
 & \quad \forall\ xs\ ::\ [t],\ x\ ::\ t\ \bullet \\
 & \qquad z\ `f`\ foldl\ g\ y\ xs \ = \ foldl\ g\ (z\ `f`\ y)\ xs \\
 & \ \Rightarrow \\
 & \qquad z\ `f`\ foldl\ g\ y\ (x : xs) \ = \ foldl\ g\ (z\ `f`\ y)\ (x : xs)
\end{array}
$$

(B).– Consideremos la función

$$
\begin{array}{lll}
foldl\ f\ z\ [\,] \quad = \ z & & --\ (1) \\
foldl\ f\ z\ (x : xs) \ = \ foldl\ f\ (f\ z\ x)\ xs & & --\ (2)
\end{array}
$$

Aplicamos el esquema anterior teniendo en cuenta que:

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

—*Caso Base*:

$$
\begin{array}{ll}
 & z\ `f`\ foldl\ g\ y\ [\,] \ = \ foldl\ g\ (z\ `f`\ y)\ [\,] \\
\equiv & \{\ 1), 1)\ \}
\end{array}
$$

$$z \text{ `}f\text{` } y \;=\; z \text{ `}f\text{` } y$$

—*Paso Inductivo*:

$$\begin{array}{ll} & z \text{ `}f\text{` } foldl\ g\ y\ (x : xs) \;=\; foldl\ g\ (z \text{ `}f\text{` } y)\ (x : xs) \\ \equiv & \{\ 2),\ 2)\ \} \\ & z \text{ `}f\text{` } foldl\ g\ (y \text{ `}g\text{` } x)\ xs \;=\; foldl\ g\ ((z \text{ `}f\text{` } y) \text{ `}g\text{` } x)\ xs \\ \Leftarrow & \{\ \text{hipótesis de inducción}\ \} \\ & foldl\ g\ (z\ 'f'\ (y \text{ `}g\text{` } x))\ xs \;=\; foldl\ g\ ((z \text{ `}f\text{` } y) \text{ `}g\text{` } x)\ xs \\ \Leftarrow & \{\ (is)\ \} \\ & z\ 'f'\ (y \text{ `}g\text{` } x) = (z \text{ `}f\text{` } y) \text{ `}g\text{` } x \\ \equiv & \{\ \text{por P1}\ \} \\ & Cierto \end{array}$$

Solución al Ejercicio 18.52 (pág. 539).– **(A)**.– Se trata de una función de plegado típica:

$$\begin{array}{ll} pliegaÁrbolG\ e\ f\ Vacío & = e \\ pliegaÁrbolG\ e\ f\ (NodoG\ x\ xs) & = f\ x\ (map\ (pliegaÁrbolG\ e\ f)\ xs) \end{array}$$

(B).–

$$pertenece\ e \;=\; pliegaÁrbolG\ False\ (\lambda\, x\ ps \;\rightarrow\; e == x \;||\; ps)$$

Solución al Ejercicio 18.53 (pág. 539).– **(A)**.– Basta con definir la lista de los numeradores y de los denominadores y emparejarlas adecuadamente:

$$\begin{array}{l} expo :: Float \;\rightarrow\; [Float] \\ expo\ x \;=\; zipWith\ (/)\ nums\ dens \\ \quad \textbf{where} \\ \quad\quad nums \;= iterate\ (*x)\ 1 \\ \quad\quad dens \;= 1 \;:\; zipWith\ (*)\ [1..]\ dens \end{array}$$

(B).– Evaluando la serie para $x \;=\; 1$ y tomando tan solo los términos requeridos, obtenemos las aproximación:

$$\begin{array}{l} aprox \;=\; sumar\ .\ takeWhile\ (\geq diezMilésima)\ \$\ expo\ 1 \\ \quad \textbf{where} \\ \quad\quad diezMilésima \;=\; 1\ /10000 \end{array}$$

Solución al Ejercicio 18.54 (pág. 540).– **(A)**.– A partir de las ecuaciones dadas para *plegar* debemos suponer que

$$plegar \;::\; t_1 \;\rightarrow\; t_2 \;\rightarrow\; t_3 \;\rightarrow\; t_4 \;\rightarrow\; t_5$$

donde $f :: t_1,\ g :: t_2,\ z :: t_3,\ \ldots$ Obviamente, dados los patrones que aparecen en el cuarto argumento, debe tenerse $t_4 \equiv Ent$. Pero, por la primera ecuación, el tipo del resultado (t_5) debe coincidir con el tipo de z. Por tanto tenemos de momento:

$$plegar :: t_1 \to t_2 \to c \to Ent \to c$$

Además, por la segunda ecuación, el tipo de f debe ser $c \to c$, e igualmente para g, de donde:

$$plegar :: (c \to c) \to (c \to c) \to c \to Ent \to c$$

(B).– Tendremos que probar, por inducción estructural sobre la estructura de Ent,

$$(*) \qquad \forall\, x, y \;.\; x, y :: Ent \;.\; (-)\; x\; y \;=\; plegar\; P\; S\; x\; y$$

Obviamente, debemos aislar una de las variables. Pero, dadas las ecuaciones, deberíamos elegir la segunda, ya que la función $(-)$ está definida vía patrones en el segundo argumento. Por tanto, probaremos

$$(*) \qquad \forall\, y,\ y :: Ent \;.\; (\forall\, x \;.\; x :: Ent \;.\; x - y \;=\; plegar\; P\; S\; x\; y)$$

por inducción sobre la variable y. Recordemos entonces el esquema de inducción, donde p representa cualquier propiedad definida sobre un dato de tipo Ent:

$$\begin{array}{ll}
 & \forall\, y,\ y :: Ent \;.\; p\; y \\
\equiv & \{\text{ esquema de inducción }\} \\
 & \quad -- \textit{Caso Base}: \\
 & p\; O \\
 & \wedge \\
 & \forall\, y,\ y :: Ent \;. \\
 & \qquad p\; y \\
 & \quad \Rightarrow \quad -- \textit{Paso Inductivo I} \\
 & \qquad p(S\; y) \\
 & \wedge \\
 & \forall\, y,\ y :: Ent \;. \\
 & \qquad p\; y \\
 & \quad \Rightarrow \quad -- \textit{Paso Inductivo II} \\
 & \qquad p(P\; y)
\end{array}$$

En nuestro caso la propiedad p es

$$p\; y \;\equiv\; (\forall\, x \;.\; x :: Ent \;.\; x - y \;=\; plegar\; P\; S\; x\; y)$$

Por tanto, la demostración tiene el siguiente aspecto:
—*Caso Base*:

$$\begin{array}{ll}
 & \forall\, x \;.\; x :: Ent \;.\; x - O \;=\; plegar\; P\; S\; x\; O \\
\equiv & \{\, 1)-,\ 1) plegar \,\} \\
 & \forall\, x \;.\; x :: Ent \;.\; x \;=\; x
\end{array}$$

$\equiv$ { cálculo de predicados }
$Cierto$

—*Paso Inductivo* I: $\forall\ y\ .\ y :: Ent\ .\ .$

$\forall\ x\ .\ x :: Ent\ .\ x\ -\ (S\ y)\ =\ plegar\ P\ S\ x\ (S\ y)$
$\equiv$ { 2)−, 2)*plegar* }
$\forall\ x\ .\ x :: Ent\ .\ P(x - y)\ =\ P(plegar\ P\ S\ x\ y)$
$\Leftarrow$ { is }
$\forall\ x\ .\ x :: Ent\ .\ x - y\ =\ plegar\ P\ S\ x\ y$
$\equiv$ { hipótesis de inducción }
$Cierto$

—*Paso Inductivo* II: $\forall\ y\ .\ y :: Ent\ .$

$\forall\ x\ .\ x :: Ent\ .\ x\ -\ (P\ y)\ =\ plegar\ P\ S\ x\ (P\ y)$
$\equiv$ { 3)−, 3)*plegar* }
$\forall\ x\ .\ x :: Ent\ .\ S(x - y)\ =\ S(plegar\ P\ S\ x\ y)$
$\Leftarrow$ { is }
$\forall\ x\ .\ x :: Ent\ .\ x - y\ =\ plegar\ P\ S\ x\ y$
$\equiv$ { hipótesis de inducción }
$Cierto$

(C).– En la ecuación (*) antes demostrada, vemos que para el cálculo de la diferencia $x - y$ a partir del valor inicial x, intercambiamos los constructores de y colocándolos en la cabeza. En forma dual podríamos encontrar la suma con un plegado. Sin embargo lo haremos en forma más genérica. Supongamos que queremos encontrar funciones y un valor inicial tales que se tenga

$$(**) \qquad \forall\ x, y\ .\ x, y :: Ent\ .\ x\ +\ y\ =\ plegar\ f\ g\ z\ x$$

La razón de que aparezca x en la expresión $plegar\ f\ g\ z\ x$ es porque, tanto la suma como $plegar$ están descritos con patrones en éste lugar. Debemos suponer además, que posiblemente f, o g o z puedan depender de y (ya que la expresión de la derecha ya depende de x). Entonces tendríamos que tener, para $x\ \equiv\ O$:

$\forall\ y\ .\ y :: Ent\ .\ O\ +\ y\ =\ plegar\ f\ g\ z\ O$
$\equiv$ { 1)+, 1)*plegar* }
$\forall\ y\ .\ y :: Ent\ .\ y\ =\ z$
$\Leftarrow$
$\forall\ y\ .\ y :: Ent\ .\ z\ \equiv\ y$

Es decir, la dependencia de y del valor z viene dada por $z\ \equiv\ y$. Por otro lado, si tomamos ahora $x\ \equiv\ S\ x'$ tendremos

$\forall\ y\ .\ y :: Ent\ .\ (S\ x')\ +\ y\ =\ plegar\ f\ g\ z\ (S\ x')$
$\equiv$ { 2)+, 2)*plegar* }
$\forall\ y\ .\ y :: Ent\ .\ S(x' + y)\ =\ f(plegar\ f\ g\ z\ x')$

$$
\begin{array}{ll}
\Leftarrow & \{ \text{ si tomamos } f \equiv S, \text{ por } is \} \\
& \forall\, y \;.\; y \;::\; Ent \;.\; x' + y \;=\; plegar\; S\; g\; z\; x'
\end{array}
$$

y vemos que obtenemos como condición la ¡hipótesis de inducción! Si razonamos igual para el predecesor, tendremos, finalmente, que se verifica

$$(**) \qquad \forall\, x, y \;.\; x, y \;::\; Ent \;.\; x \;+\; y \;=\; plegar\; S\; P\; y\; x$$

y lo anterior no es necesario demostrarlo ya que hemos construido la expresión $plegar\; S\; P\; y\; x$ ¡por inducción! Un ejemplo puede ser:

$$
\begin{array}{l}
\textsc{Main}\!> \; dos \\
S\;(S\;O) \;::\; Ent
\end{array}
$$

$$
\begin{array}{l}
\textsc{Main}\!> \; mTres \\
P\;(P\;(P\;O)) \;::\; Ent
\end{array}
$$

$$
\begin{array}{l}
\textsc{Main}\!> \; dos \;+\; mTres\; \mathbf{where}\; x \;+\; y \;=\; plegar\; S\; P\; y\; x \\
S\;(S\;(P\;(P\;(P\;O)))) \;::\; Ent
\end{array}
$$

$$
\begin{array}{l}
\textsc{Main}\!> \; \mathbf{let}\; x \;+\; y \;=\; plegar\; S\; P\; y\; x\; \mathbf{in}\; dos \;+\; dos \\
S\;(S\;(S\;(S\;O))) \;::\; Ent
\end{array}
$$

Obsérvese que ¡no se simplifican las expresiones!, aunque $2 - 3 = -1$.

Busquemos ahora una función $aInt \;::\; Ent \;\rightarrow\; Int$ tal que

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

Supongamos además que queremos definirla a partir de un plegado; entonces debemos establecer antes su comportamiento, que podemos dar en forma de código

$$
\begin{array}{lcl}
aInt\;O & = & 0 \\
aInt\;(S\;x) & = & aInt\;x \;+\; 1 \\
aInt\;(P\;x) & = & aInt\;x \;-\; 1
\end{array}
$$

de forma que tal función establece un isomorfismo entre Ent y Int. A partir de su comportamiento, podemos buscar las funciones f y g, y el valor de z tales que se tenga

$$(1) \qquad \forall\, x \;.\; x \;::\; Ent \;.\; aInt\;x \;=\; plegar\; f\; g\; z\; x$$

Entonces, razonamos igual que antes. Si tomamos $x \equiv\; O$ debería tenerse $z \equiv\; 0$ (atención, 0 es el cero de Int, mientras que O es el cero de Ent). Por otro lado, si tomamos $P\;x$,

$$
\begin{array}{ll}
& \forall\, x \;.\; x \;::\; Ent \;.\; aInt\;(P\;x) \;=\; plegar\; f\; g\; 0\;(P\;x) \\
\equiv\!\equiv & \{\, 3)aInt,\, 3)plegar \,\} \\
& \forall\, x \;.\; x \;::\; Ent \;.\; aInt\;x \;-\; 1 \;=\; g(plegar\; f\; g\; 0\; x) \\
\Leftarrow & \{ \text{ si tomamos } g \equiv \lambda y \rightarrow y - 1, \text{ por } is \} \\
& \forall\, x \;.\; x \;::\; Ent \;.\; aInt \;=\; plegar\; g\; z\; x
\end{array}
$$

y de nuevo obtenemos ¡la hipótesis de inducción! Razonando igual para el sucesor obtenemos finalmente

$$(1) \qquad \forall\, x \;.\; x :: Ent \;.\; aInt\; x = plegar\; (1+)\; (flip\; (-)\; 1)\; 0$$

Recuerde que $(+1)$ puede ser una abreviatura de $\lambda\, y \to y + 1$, pero (-1) no es una forma abreviada de la expresión $\lambda\, y \to y - 1$:

```
MAIN> (+1) 3
4 :: Integer

MAIN> (-1) 3
ERROR : Illegal Haskell 98 class constraint in inferred type ...

MAIN> aInt tres where aInt = plegar (+1) (flip (-) 1) 0
3 :: Integer

MAIN> let aInt = plegar (+1) (flip (-) 1) 0 in aInt mTres
-3 :: Integer
```

Solución al Ejercicio 18.55 (pág. 540).–

```
m :: Int
m = 5
type Contador = [Int]
contadorInicial :: Contador
```

(A).– $contadorInicial = take\; (2 * m + 1)\; (copy\; 0)$ **where** $copy\; a = a : copy\; a$

(B).–

```
type Posicion = Int
incrementa :: Posicion -> Contador -> Contador
incrementa 0       (c : cs) = (c + 1) : cs
incrementa (n + 1) (c : cs) = c : incrementa n cs
```

(C).–

```
type Semilla = Int
alea         :: Semilla -> [Int]
alea sem     = iterate (\x -> (77 * x + 1) `rem` 1024) sem
dados        :: Int -> [Int]
dados sem    = map (\y -> ((m + 1) * y) `div` 1023) (alea sem)
```

(D).–

```
type NúmeroDeTiradas = Int
simula          :: NúmeroDeTiradas -> Semilla -> Contador
simula n sem    = tira n contadorInicial (dados sem)
tira 0       con _           = con
tira (n + 1) con (d : d' : ds) = tira n (incrementa (d + d') con) ds
```

(E).–

```
estudioEstadístico sem n =
        (simula n sem, [ (realToFrac n) * frec v |v ← [0..2 * m]] )
```

Con la función anterior tendremos el siguiente diálogo:

```
MAIN> estudioEstadístico 43 100
([3, 11, 7, 13, 12, 12, 12, 6, 14, 7, 3],
[2.77778, 5.55556, 8.33333, 11.1111, 13.8889, 16.6667,
13.8889, 11.1111, 8.33333, 5.55556, 2.77778]) :: (Contador, [Double])
```

Otra posibilidad es cambiar algo las presentaciones de las frecuencias teóricas para que aparezcan como números fraccionarios. Entonces realizamos algunos cambios sobre las conversiones de tipos de las operaciones:

```
frec2 :: Int → Ratio Integer
frec2 v
  | v < m     = toRational (v + 1)/ toRational ((m + 1) ↑ 2)
  | otherwise = toRational (2 * m − v + 1)/ toRational ((m + 1) ↑ 2)

estudioEstadístico2 :: NúmeroDeTiradas → Semilla →
                       (Contador, [Ratio Integer])
estudioEstadístico2 sem n =
  (simula n sem, [ toRational n * frec2 v | v ← [0..2 * m]] )
```

Por ejemplo, tenemos la siguiente simulación:

```
MAIN> estudioEstadístico2 43 100
([3, 11, 7, 13, 12, 12, 12, 6, 14, 7, 3],
[25 % 9, 50 % 9, 25 % 3, 100 % 9, 125 % 9, 50 % 3,
125 % 9, 100 % 9, 25 % 3, 50 % 9, 25 % 9]) :: (Contador, [Ratio Integer])
```

donde observamos los cocientes exactos para las frecuencias: 24 % 9 (veinticuatro novenos), etc.

Finalmente daremos una tercera solución: que la semilla inicial sea elegida por el sistema. En ese caso podemos utilizar una función del módulo *Random* para calcular un número aleatorio dentro de un intervalo:

```
module Random ( ... , Random( randomR, ... ), getStdRandom, ... )
 where
  ...
    randomR :: RandomGen g ⇒ (a, a) → g → (a, g)
  ...
  getStdRandom :: (StdGen → (a, StdGen)) → IO a
  ...
```

La función *randomR* (que aparece en una clase dentro del módulo, aunque lo importante es que existen instancias para los tipos más utilizados) toma un intervalo y devuelve

una función que generará una semilla. Tal función puede ser argumento de la función $getStdRamdom$, que devuelve en forma monádica la semilla. Así, podemos escribir, en nuestro programa

$$
\begin{array}{l}
\textbf{import}\ Random \\
\ldots \\
simulaIO\ ::\ NúmeroDeTiradas\ \rightarrow\ IO\ () \\
simulaIO\ n\ = getStdRandom\ (randomR\ (1, 1024))\ >>=\ \lambda\, sem\ \rightarrow \\
\qquad\qquad print\ (show\ (estudioEstadístico\ sem\ n))
\end{array}
$$

Observa que el resultado final de $simulaIO$ es un valor monádico. Un ejemplo de diálogo puede ser, para $m = 3$,

```
MAIN> simulaIO 300
"([19,37,56,72,53,41,22],[18.75,37.5,56.25,75.0,56.25,37.5,18.75])" :: IO ()
```

Solución al Ejercicio 18.56 (pág. 542).– Véase también la solución del Ejercicio 18.54 (pág. 540), pero observa que el operador $(+)$ aparece definido por patrones en el segundo argumento; por tanto habrá ligeros cambios.
(A).–

$$
\begin{array}{l}
\quad \forall\, x,\ y\ ::\ Ent\ \centerdot\ x\ -\ y\ =\ x\ +\ negar\ y \\
\equiv \\
\qquad \textit{-- Caso Base} \\
\quad \forall\, x\ ::\ Ent\ \centerdot\ x\ -\ O\ =\ x\ +\ negar\ O \\
\ \wedge \\
\qquad \textit{-- Paso Inductivo I} \\
\quad \forall\, y\ ::\ Ent\ \centerdot \\
\qquad\quad \forall\, x\ ::\ Ent\ \centerdot\ x\ -\ y\ =\ x\ +\ negar\ y \\
\quad \Rightarrow \\
\qquad\quad \forall\, x\ ::\ Ent\ \centerdot\ x\ -\ S\ y\ =\ x\ +\ negar\ (S\ y) \\
\ \wedge \\
\qquad \textit{-- Paso Inductivo II} \\
\quad \forall\, y\ ::\ Ent\ \centerdot \\
\qquad\quad \forall\, x\ ::\ Ent\ \centerdot\ x\ -\ y\ =\ x\ +\ negar\ y \\
\quad \Rightarrow \\
\qquad\quad \forall\, x\ ::\ Ent\ \centerdot\ x\ -\ P\ y\ =\ x\ +\ negar\ (P\ y)
\end{array}
$$

(B).–

$$
\begin{array}{l}
\quad x\ -\ O\ =\ x\ +\ negar\ O \\
\equiv\ \{\, 1)-,\ 1)negar\,\} \\
\quad O\ =\ x\ +\ O \\
\equiv\ \{\, 1)+\,\} \\
\quad O\ =\ O
\end{array}
$$

$$
x\ -\ S\ y\ =\ x\ +\ negar\ (S\ y)
$$

$$
\begin{array}{ll}
\equiv & \{\, 2)-,\, 2)negar \,\} \\
& P\,(x\ -\ y)\ =\ x\ +\ P\,(negar\ y) \\
\equiv & \{\, 2)+ \,\} \\
& P\,(x\ -\ y)\ =\ P\,(x\ +\ negar\ y) \\
\Leftarrow & \{\ is \,\} \\
& x\ -\ y\ =\ x\ +\ negar\ y \\
\Leftarrow & \{\ \text{hipótesis de inducción} \,\} \\
& Cierto
\end{array}
$$

$$
\begin{array}{ll}
& x\ -\ P\ y\ =\ x\ +\ negar\ (P\ y) \\
\equiv & \{\, 3)-,\, 3)negar \,\} \\
& S\,(x\ -\ y)\ =\ x\ +\ S\,(negar\ y) \\
\equiv & \{\, 3)+ \,\} \\
& S\,(x\ -\ y)\ =\ S\,(x\ +\ negar\ y) \\
\Leftarrow & \{\ is \,\} \\
& x\ -\ y\ =\ x\ +\ negar\ y \\
\Leftarrow & \{\ \text{hipótesis de inducción} \,\} \\
& Cierto
\end{array}
$$

(C).– Razonamos igual que en la solución del Ejercicio 18.54 (pág. 540), pero al aparecer los patrones a la derecha habrá algún cambio. Por ejemplo, si queremos buscar funciones f y g, así como un valor de z tales que se tenga

$$(1) \qquad \forall\, x, y\ \centerdot\ x, y\ ::\ Ent\ \centerdot\ x\ +\ y\ =\ plegar\ f\ g\ z\ y$$

aparece y en la expresión $plegar\ f\ g\ z\ y$ porque la suma está descrita con patrones en este lugar. De nuevo debemos suponer que posiblemente f, o g o z puedan depender de x (ya que la expresión de la derecha ya depende de y). Entonces tendríamos que tener, para $y\ \equiv\ O$:

$$
\begin{array}{ll}
& \forall\, x\ \centerdot\ x\ ::\ Ent\ \centerdot\ x\ +\ O\ =\ plegar\ f\ g\ z\ O \\
\equiv & \{\, 1)+,\, 1)plegar \,\} \\
& \forall\, x\ \centerdot\ x\ ::\ Ent\ \centerdot\ x\ =\ z \\
\Leftarrow & \\
& \forall\, x\ \centerdot\ x\ ::\ Ent\ \centerdot\ z\ \equiv\ x
\end{array}
$$

Es decir, la dependencia de x del valor z viene dada por: $z\ \equiv\ x$. Por otro lado, si tomamos ahora $y\ \equiv\ S\ y'$, tendremos

$$
\begin{array}{ll}
& \forall\, x\ \centerdot\ x\ ::\ Ent\ \centerdot\ x\ +\ (S\ y')\ =\ plegar\ f\ g\ z\ (S\ y') \\
\equiv & \{\, 2)+,\, 2)plegar \,\} \\
& \forall\, x\ \centerdot\ x\ ::\ Ent\ \centerdot\ S(x + y')\ =\ f(plegar\ f\ g\ z\ y') \\
\Leftarrow & \{\ \text{si tomamos } f \equiv S, \text{ por } is \,\} \\
& \forall\, x\ \centerdot\ x\ ::\ Ent\ \centerdot\ x + y'\ =\ plegar\ S\ g\ z\ y'
\end{array}
$$

y vemos que obtenemos como condición ¡la hipótesis de inducción! Si razonamos igual para el predecesor, tendremos finalmente que se verifica

$$(1) \qquad \forall\, x, y \;\centerdot\; x, y \;::\; Ent \;\centerdot\; x + y = plegar\ S\ P\ x\ y$$

y de nuevo lo anterior no es necesario demostrarlo ya que hemos construido la expresión $plegar\ S\ P\ x\ y$ ¡por inducción!

Si razonamos de la misma forma con la diferencia

$$(2) \qquad \forall\, x, y \;\centerdot\; x, y \;::\; Ent \;\centerdot\; x - y = plegar\ f\ g\ z\ y$$

aparece de nuevo y en la expresión $plegar\ f\ g\ z\ y$ porque la diferencia está descrita con patrones en el mismo lugar (a la derecha del operador). Tomando $y \equiv O$ encontramos:

$$\begin{array}{ll} & \forall\, x \;\centerdot\; x \;::\; Ent \;\centerdot\; x - O = plegar\ f\ g\ z\ O \\ \equiv\equiv & \{\ 1)-,\ 1)plegar\ \} \\ & \forall\, x \;\centerdot\; x \;::\; Ent \;\centerdot\; x = z \\ \Leftarrow & \\ & \forall\, x \;\centerdot\; x \;::\; Ent \;\centerdot\; z \equiv x \end{array}$$

O sea: $z \equiv x$. Por otro lado, si tomamos $y \equiv S\ y'$, tendremos

$$\begin{array}{ll} & \forall\, x \;\centerdot\; x \;::\; Ent \;\centerdot\; x - (S\ y') = plegar\ f\ g\ z\ (S\ y') \\ \equiv\equiv & \{\ 2)-,\ 2)plegar\ \} \\ & \forall\, x \;\centerdot\; x \;::\; Ent \;\centerdot\; P(x - y') = f(plegar\ f\ g\ z\ y') \\ \Leftarrow & \{\ \text{si tomamos } f \equiv P, \text{ por } is\ \} \\ & \forall\, x \;\centerdot\; x \;::\; Ent \;\centerdot\; x - y' = plegar\ P\ g\ z\ y' \end{array}$$

de donde obtenemos ¡la hipótesis de inducción!, y razonando igual para el predecesor encontramos

$$(2) \qquad \forall\, x, y \;\centerdot\; x, y \;::\; Ent \;\centerdot\; x - y = plegar\ P\ S\ x\ y$$

Por consiguiente, la definición de instancia a partir del plegado podría haber sido:

$$\begin{array}{l} \textbf{instance}\ Num\ Ent\ \textbf{where} \\ \quad (+)\ x = plegar\ S\ P\ x \\ \quad (-)\ x = plegar\ P\ S\ x \end{array}$$

Observe que el diálogo con respecto al ejercicio anteriormente citado cambia ligeramente:

$$\begin{array}{l} \text{MAIN}{>}\ dos \\ S\ (S\ O)\ ::\ Ent \end{array}$$

$$\begin{array}{l} \text{MAIN}{>}\ mTres \\ P\ (P\ (P\ O))\ ::\ Ent \end{array}$$

$$\begin{array}{l} \text{MAIN}{>}\ dos\ +\ mTres\ \textbf{where}\ (+)\ x = plegar\ S\ P\ x \\ P\ (P\ (P\ (S\ (S\ O))))\ ::\ Ent \end{array}$$

dado que antes aparecía en la última línea $S\ (S\ (P\ (P\ (P\ O))))$.

(D).– Para buscar la función producto, razonamos igual que en la solución del Ejercicio 18.54 (pág. 540). Es decir, partimos de la especificación del producto

$$\begin{array}{l}\textbf{instance}\ \mathit{Num}\ \mathit{Ent}\ \textbf{where}\\ \quad x * O \ = \ O\\ \quad x * (S\ y) \ = \ x * y \ + \ x\\ \quad x * (P\ y) \ = \ x * y \ - \ x\end{array}$$

Si buscamos verificar

$$(3) \qquad \forall\, x, y \ \centerdot\ x, y \ ::\ \mathit{Ent} \ \centerdot\ x * y \ = \ \mathit{plegar}\ f\ g\ z\ y$$

tomando $y \ \equiv \ O$ encontramos $z \equiv \ O$. Si tomamos $S\ y$, tendremos

$$\begin{array}{ll} & \forall\, x \ \centerdot\ x \ ::\ \mathit{Ent} \ \centerdot\ x * (S\ y) \ = \ \mathit{plegar}\ f\ g\ z\ (S\ y)\\ \equiv\equiv & \{\ 2)*,\ 2)\mathit{plegar}\ \}\\ & \forall\, x \ \centerdot\ x \ ::\ \mathit{Ent}\ x * y \ + \ x \ = \ f(\mathit{plegar}\ f\ g\ z\ y)\\ \Leftarrow & \{\ \text{si tomamos } f\ u = u + x, \text{ por } \mathit{is}\ \}\\ & \forall\, x \ \centerdot\ x \ ::\ \mathit{Ent} \ \centerdot\ x * y \ + \ x \ = \ (\mathit{plegar}\ f\ g\ z\ y) \ + \ x\end{array}$$

de donde obtenemos ¡la hipótesis de inducción!, y razonando igual para el predecesor encontramos

$$(3) \qquad \forall\, x, y \ \centerdot\ x, y \ ::\ \mathit{Ent} \ \centerdot\ x * y \ = \ \mathit{plegar}\ (+x)\ (\mathit{flip}\ (-)\ x)\ O\ y$$

(Atención, la expresión $(-\ x)$ no es una función válida). Realmente, también podemos sustituir la expresión $\mathit{flip}\ (-)\ x$ por $\mathit{negate}\ x$. Por consiguiente, la definición de instancia a partir del plegado podría haber sido:

$$\begin{array}{l}\textbf{instance}\ \mathit{Num}\ \mathit{Ent}\ \textbf{where}\\ \quad (*)\ x \ = \ \mathit{plegar}\ (+x)\ (\mathit{flip}\ (-)\ x)\ O\end{array}$$

$$\begin{array}{l}\textsc{Main}\!>\ \mathit{dos}\\ S\ (S\ O)\ ::\ \mathit{Ent}\end{array}$$

$$\begin{array}{l}\textsc{Main}\!>\ \mathit{mTres}\\ P\ (P\ (P\ O))\ ::\ \mathit{Ent}\end{array}$$

$$\begin{array}{l}\textsc{Main}\!>\ \mathit{dos} * \mathit{mTres}\ \textbf{where}\ (*)\ x \ = \ \mathit{plegar}\ (+x)\ (\mathit{flip}\ (-)\ x)\ O\\ P\ (P\ (P\ (P\ (P\ O))))\ ::\ \mathit{Ent}\end{array}$$

$$\begin{array}{l}\textsc{Main}\!>\ \mathit{dos} * \mathit{mTres}\ \textbf{where}\ (*)\ x \ = \ \mathit{plegar}\ (+x)\ (-x)\ O\\ \mathit{ERROR}:\ \mathit{Type\ error\ in\ application}\ \ldots\end{array}$$

Solución al Ejercicio 18.57 (pág. 542).– Asignamos tipos a argumentos y resultado tendremos

$$y :: t1, \quad z :: t2, \quad w :: t3, \quad (w\ z)\ (z\ y)\ ::\ t4, \quad x\ ::\ t1 \ \rightarrow\ t2 \ \rightarrow\ t3 \ \rightarrow\ t4$$

Pero entonces

$$
\begin{array}{l}
\quad (w\ z)\ (z\ y)\ ::\ t4 \\
\Rightarrow \\
\quad z\ y\ ::\ t5,\ w\ z\ ::\ t5\ \rightarrow\ t4 \\
\\
\quad z\ y\ ::\ t5 \\
\Rightarrow \\
\quad y ::\ t1,\ z\ ::\ t1\ \rightarrow\ t5 \\
\Rightarrow \\
\quad t2\ =\ t1 \rightarrow\ t5 \\
\\
\quad w\ z\ ::\ t5\ \rightarrow\ t4 \\
\Rightarrow \\
\quad z\ ::\ t1\ \rightarrow\ t5,\ w\ ::(t1\ \rightarrow\ t5) \rightarrow\ t5\ \rightarrow\ t4
\end{array}
$$

y tendremos

$$x\ ::\ a\ \rightarrow\ (a\ \rightarrow\ b)\ \rightarrow\ ((a\ \rightarrow\ b)\ \rightarrow\ b\ \rightarrow\ c)\ \rightarrow\ c$$

haciendo los siguientes renombramientos:

$t1$	a
$t2 = t1 \rightarrow t5$	$a \rightarrow b$
$t3 = (t1 \rightarrow t5) \rightarrow t5 \rightarrow t4$	$(a \rightarrow b) \rightarrow b \rightarrow c$
$t4$	c
$t5$	b

Solución al Ejercicio 18.58 (pág. 543).– **(A)**.– Ver Figura 19.4:

$$s\ =\ (0,1) : zipWith\ (\lambda\ (x,y)\ z\ \rightarrow\ (x+z, y*z))\ s\ [1..]$$

(B).–

$$t\ =\ concat\ (map\ (\lambda\ (x,y)\ \rightarrow\ [(x,y),(y,x)])\ s)$$

– – o bien en la forma

$$t\ =\ t'\ s\ \mathbf{where}\ t'\ ((x,y) : xs)\ =\ (x,y) : (y,x) : t'\ xs$$

Solución al Ejercicio 18.59 (pág. 543).– **(A)**.– La siguiente función crea un diccionario vacío:

$$
\begin{array}{l}
creaDicc\ ::\ Dicc\ a \\
creaDicc\ =\ [\,]
\end{array}
$$

Para actualizar un diccionario por su clave y valor: si la clave no existe, se inserta con el valor; en otro caso se cambia al nuevo valor. Las claves deben estar ordenadas de menor a mayor:

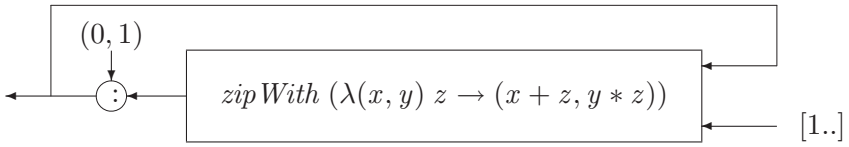

Figura 19.4: Red de procesos del Ejercicio 18.58..

```
actualiza :: Dicc a → String → a → Dicc a
actualiza []              c v = [A c v]
actualiza ((A c' v') : xs) c v | c' == c     = (A c v) : xs
                               | c' > c      = (A c v) : (A c' v') : xs
                               | otherwise   = (A c' v') : actualiza xs c v
```

El siguiente predicado devuelve *True* si la clave se encuentra en el diccionario. Debe tenerse en cuenta que las claves están ordenadas:

```
está :: Dicc a → String → Bool
está []              c = False
está ((A c' v') : xs) c | c' == c     = True
                        | c' > c      = False
                        | otherwise   = está xs c
```

La siguiente función devuelve el valor asociado a una clave o un error si no está. De nuevo debe tenerse en cuenta que las claves están ordenadas:

```
valor :: Dicc a → String → a
valor []              c = error "La clave no existe"
valor ((A c' v') : xs) c | c' == c     = v'
                         | c' > c      = error "La clave no existe"
                         | otherwise   = valor xs c
```

(B).– La siguiente función, a partir de una lista de palabras (donde posiblemente hay palabras repetidas) cuenta las veces que aparece cada palabra, vía un diccionario:

```
cuentaPalabras :: [String] → Dicc Int
cuentaPalabras = foldr incrementa creaDicc
  where incrementa p d | está d p    = actualiza d p (n + 1)
                       | otherwise   = actualiza d p 1
                       where n       = valor d p
```

(C).–

```
porApariciones :: [String] → [(String, Int)]
porApariciones = aListaPares . ordenaApa . cuentaPalabras
```

```
ordenaApa []                 = []
ordenaApa [x]                = [x]
ordenaApa ((A c v) : xs) =
      ordenaApa [A c' v' | A c' v' ← xs, v' >= v] ++
                (A c v) : ordenaApa [A c' v' | A c' v' ← xs, v' < v]

aListaPares = map (λ(A c v) → (c, v))
```

(D).– La función *acumuladora* sobre un diccionario puede ser

```
acum :: Encuesta → Dicc (Dicc Int) → Dicc (Dicc Int)
acum e dicc = foldr acumula dicc e
  where acumula (p, c) d
          | está d p    = actualiza d p (actualiza di c (n + 1))
          | otherwise = actualiza d p (actualiza creaDicc c 1)
          where
                  di = valor d p
                  n  = if está di c then valor di c else 0
```

Finalmente, tendremos que definir una función tal que a partir de una lista de encuestas produzca como resultado un diccionario, donde, para cada pregunta de la encuesta, aparezca otro diccionario en el que, asociada a cada respuesta, aparezca el número de veces que ésta se produce. Tal función puede ser:

```
resultados :: [Encuesta] → Dicc (Dicc Int)
resultados = foldr acum creaDicc
```

Solución al Ejercicio 18.60 (pág. 545).– **(A)**.–

```
intercalar x []       = []
intercalar x [y]      = [y]
intercalar x (y : ys) = y : x : intercalar x ys
```

(B).–

```
pinta :: Imagen → String
pinta = concat . intercalar "\n" . map (mapsustituir)
  where
    sustituir 0 = '.'
    sustituir 1 = 'X'
    sustituir _ = error "pixel erróneo"


refH, refV, refHV :: Imagen → Imagen
refH  = map reverse
refV  = reverse
refHV = refH . refV
```

(C).–

```
(>+<) :: Imagen → Imagen → Imagen
(>+<) = (++)

(>-<) :: Imagen → Imagen → Imagen
(>-<) = zipWith (++)
```

(D).–

```
negar :: Imagen → Imagen
negar = map (map inv)
  where
    inv ch = if ch == 1 then 0 else 1
```

(E).–

```
superponer :: Imagen → Imagen → Imagen
superponer = zipWith (zipWith (<+>))
  where
    1 <+> 1 = 1
    x <+> y = x + y
repite :: Int → [a] → [a]
repite n = foldr (λ x xs → [x| _ ← [1..n]] ++ xs) []

zoom :: Int → Imagen → Imagen
zoom n = repite n . map (repite n)
```

Solución al Ejercicio 18.61 (pág. 548).– Ver también soluciones del Ejercicio 18.54 (pág. 540) y Ejercicio 18.56 (pág. 542).
(A).– Dado que

```
plegar :: (a → a) → (a → a) → a → Ent → a
```

tendremos que para las ecuaciones:

```
f1 = plegar not not True
f2 = plegar not not False
f3 = plegar S S O
```

los tipos son

```
f1 :: Ent → Bool   -- a sería Bool
f2 :: Ent → Bool   -- a sería Bool
f3 :: Ent → Ent    -- a sería Ent
```

mientras que por ejemplo, para la función de ecuación $g = plegar\ not\ not$ tendríamos $g :: Bool \to Ent \to Bool$. Por tanto $f1 = g\ True$ y $f2 = g\ False$.

(B).– $f1\ e$ evalúa el predicado e *es un entero par*, mientras que $f2\ e$ evalúa el predicado e *es un entero impar*. Obsérvese que $f3$ sustituye todos los constructores por el constructor S, por lo que si el entero está normalizado (o sea, es de la forma $S\ (\ S\ (\ \ldots\ (S\ O)\ldots\ )$), o bien de la forma $P\ (\ P\ (\ \ldots\ (P\ O)\ldots\ )$) entonces $f3$ coincide con la función valor absoluto. En cualquier caso, veremos que $f3$ no altera la paridad de su argumento.
(C).–

$$
\begin{array}{ll}
 & \forall\ y,\ y :: Ent\ \centerdot\ f1\ y\ =\ f1\ (f3\ y) \\
\equiv & \{\ \text{esquema de inducción}\ \} \\
 & \quad -- \textit{Caso Base}: \\
 & f1\ O\ =\ f1\ (f3\ O) \\
\wedge & \\
 & \forall\ y,\ y :: Ent\ \centerdot \\
 & \qquad f1\ y\ =\ f1\ (f3\ y) \\
 & \Rightarrow\ \ -- \textit{Paso Inductivo} \text{ I} \\
 & \qquad f1\ (S\ y)\ =\ f1\ (f3\ (S\ y)) \\
\wedge & \\
 & \forall\ y,\ y :: Ent\ \centerdot \\
 & \qquad f1\ y\ =\ f1\ (f3\ y) \\
 & \Rightarrow\ \ -- \textit{Paso Inductivo} \text{ II} \\
 & \qquad f1\ (P\ y)\ =\ f1\ (f3\ (P\ y))
\end{array}
$$

—*Caso Base*:

$$
\begin{array}{ll}
 & f1\ O\ =\ f1\ (f3\ O) \\
\equiv & \{\ f1, f3\ \} \\
 & plegar\ not\ not\ True\ O\ =\ f1(plegar\ S\ S\ O\ O) \\
\equiv & \{\ 1)plegar\ \} \\
 & True\ =\ f1\ O \\
\equiv & \{\ f1\ \} \\
 & True\ =\ True
\end{array}
$$

—*Paso Inductivo* I:

$$
\begin{array}{ll}
 & f1\ (S\ y)\ =\ f1\ (f3\ (S\ y)) \\
\equiv & \{\ f1, f3\ \} \\
 & plegar\ not\ not\ True\ (S\ y)\ =\ f1\ (plegar\ S\ S\ O\ (S\ y)) \\
\equiv & \{\ 2)plegar\ \} \\
 & not(plegar\ not\ not\ True\ y)\ =\ f1\ (\ S(plegar\ S\ S\ O\ y)) \\
\equiv & \{\ f1\ \} \\
 & not(plegar\ not\ not\ True\ y)\ =\ plegar\ not\ not\ True\ (S(plegar\ S\ S\ O\ y)) \\
\equiv & \{\ 2)plegar\ \} \\
 & not(plegar\ not\ not\ True\ y)\ =\ not\ (plegar\ not\ not\ True\ (plegar\ S\ S\ O\ y) \\
\equiv & \{\ f1, f3\ \} \\
 & not(f1\ y)\ =\ not\ (f1\ (f3\ y)) \\
\Leftarrow & \{\ is\ \} \\
 & f1\ y\ =\ f1\ (f3\ y)
\end{array}
$$

$\equiv$ { hipótesis de inducción }
$Cierto$

El *Paso Inductivo* II es similar. La interpretación de la propiedad es que $f3$ no altera la paridad. Si e está normalizado, entonces $f3\ e \equiv valorAbsoluto\ e$ y la propiedad dice que si e es par, entonces también es par su valor absoluto.

Veamos ahora la propiedad

$$(3) \qquad \forall\ y,\ y :: Ent\ \centerdot\ f1\ y\ =\ not\ (f2\ y)$$

El esquema sería en este caso

$$\begin{array}{l}
\quad \forall\ y,\ y :: Ent\ \centerdot\ f1\ y\ =\ not\ (f2\ y) \\
\equiv \quad \{\text{esquema de inducción}\} \\
\quad\ \text{-- } \textit{Caso Base}\text{:} \\
\quad f1\ O\ =\ not\ (f2\ O) \\
\wedge \\
\quad \forall\ y,\ y :: Ent\ \centerdot \\
\qquad\quad f1\ y\ =\ not\ (f2\ y) \\
\qquad \Rightarrow \quad \text{-- } \textit{Paso Inductivo} \text{ I} \\
\qquad\quad f1\ (S\ y)\ =\ not\ (f2\ (S\ y)) \\
\wedge \\
\quad \forall\ y,\ y :: Ent\ \centerdot \\
\qquad\quad f1\ y\ =\ not\ (f2\ y) \\
\qquad \Rightarrow \quad \text{-- } \textit{Paso Inductivo} \text{ II} \\
\qquad\quad f1\ (P\ y)\ =\ not\ (f2\ (P\ y))
\end{array}$$

—*Caso Base*:

$$\begin{array}{l}
\quad f1\ O\ =\ not\ (f2\ O) \\
\equiv \quad \{f1, f3\} \\
\quad plegar\ not\ not\ True\ O\ =\ not\ (plegar\ not\ not\ False\ O) \\
\equiv \quad \{1)plegar\} \\
\quad True\ =\ not\ False \\
\equiv \\
\quad Cierto
\end{array}$$

—*Paso Inductivo* I:

$$\begin{array}{l}
\quad f1\ (S\ y)\ =\ not\ (f2\ (S\ y)) \\
\equiv \quad \{f1, f3\} \\
\quad plegar\ not\ not\ True\ (S\ y)\ =\ not\ (plegar\ not\ not\ False\ (S\ y)) \\
\equiv \quad \{2)plegar\} \\
\quad not(plegar\ not\ not\ True\ y)\ =\ not\ (\ not\ (plegar\ not\ not\ False\ y)) \\
\Leftarrow \quad \{\ is\} \\
\quad plegar\ not\ not\ True\ y\ =\ not\ (plegar\ not\ not\ False\ y) \\
\equiv \quad \{f1, f3\} \\
\quad f1\ y\ =\ not(f2\ y)
\end{array}$$

$\equiv$ { hipótesis de inducción }
$Cierto$

—*Paso Inductivo* II:

$f1\ (P\ y)\ =\ not\ (f2\ (P\ y))$
$\equiv$ $\{f1, f3\}$
$plegar\ not\ not\ True\ (P\ y)\ =\ not\ (plegar\ not\ not\ False\ (P\ y))$
$\equiv$ { 2)*plegar* }
$not(plegar\ not\ not\ True\ y)\ =\ not\ (\ not\ (plegar\ not\ not\ False\ y))$
$\equiv$ { hipótesis de inducción }
$Cierto$

La interpretación de la propiedad (3) es: $par\ \equiv\ not\ .\ impar$.
(D).– La propiedad que se nos pide es

$$(4) \qquad \forall\ y\ \centerdot\ y :: Ent\ \centerdot\ S\ O\ *\ y\ =\ y$$

Pero el producto puede definirse de varias formas, como puede verse en la solución del Ejercicio 18.56 (pág. 542). Si la definición fuera

$$x\ *\ y\ =\ plegar\ (+y)\ (flip\ (-)\ y)\ O\ x$$

entonces la demostración es inmediata

$S\ O\ *\ y$
$\equiv$ { definición de $*$ }
$plegar\ (+y)\ (flip\ (-)\ y)\ O\ (S\ O)$
$\equiv$ { 2)*plegar* }
$(+y)\ (plegar\ (+y)\ (flip\ (-)\ y)\ O\ O)$
$\equiv$ { 1)*plegar* }
$(+y)\ O$
$\equiv$ { 1)+ }
y

pero si la propiedad que se nos pide fuera

$$\forall\ x\ \centerdot\ x :: Ent\ \centerdot\ x\ *\ S\ O\ =\ x$$

entonces la prueba no sería directa.

En forma dual, si la definición del producto fuera

$$x\ *\ y\ =\ plegar\ (+x)\ (flip\ (-)\ x)\ O\ y$$

entonces la demostración no solamente es más complicada, sino que es imposible. Ya que en la propiedad (4) interviene y a la derecha de $*$, así como en la definición del producto aparece y como el último argumento de la función $plegar$, podemos pensar en una estrategia por inducción. Aplicamos entonces el esquema de inducción sobre y.

El caso base es trivial. Para intentar los pasos inductivos nos interesa comprobar que la función $*$ verifica las propiedades

$$(4.1) \qquad \forall\, x, y \;.\; x, y :: \mathit{Ent} \;.\; x * (S\ y) = x * y + x$$
$$(4.2) \qquad \forall\, x, y \;.\; x, y :: \mathit{Ent} \;.\; x * (P\ y) = x * y - x$$

Ambas se prueban fácilmente, bien a través de la propiedad universal, o bien directamente. Veamos solo la primera, $\forall\, x, y \;.\; x, y :: \mathit{Ent}\;.$

$$\begin{array}{ll} & x * (S\ y) \\ \equiv & \{\text{ definición de } * \} \\ & \mathit{plegar}\ (+x)\ (\mathit{flip}\ (-)\ x)\ O\ (S\ y) \\ \equiv & \{\ 2)\mathit{plegar}\ \} \\ & (+x)\ (\ \mathit{plegar}\ (+x)\ (\mathit{flip}\ (-)\ x)\ O\ y) \\ \equiv & \{\text{ definición de } * \} \\ & (+x)\ (\ x * y\ ) \\ \equiv & \\ & x * y + x \end{array}$$

Una vez probadas las propiedades (4.1) y (4.2), al intentar los pasos inductivos nos encontramos con algún problema:
—*Paso Inductivo* I:

$$\begin{array}{ll} & S\ O * (S\ y) = S\ y \\ \equiv & \{\ (4.1)\ \} \\ & S\ O * y + S\ O = S\ y \\ \equiv & \{\text{ hipótesis de inducción }\} \\ & y + S\ O = S\ y \end{array}$$

—*Paso Inductivo* II:

$$\begin{array}{ll} & S\ O * (P\ y) = P\ y \\ \equiv & \{\ (4.2)\ \} \\ & S\ O * y - S\ O = P\ y \\ \equiv & \{\text{ hipótesis de inducción }\} \\ & y - S\ O = P\ y \end{array}$$

En definitiva, necesitamos las propiedades

$$(4.3) \qquad \forall\, y \;.\; y :: \mathit{Ent} \;.\; y + S\ O = S\ y$$
$$(4.4) \qquad \forall\, y \;.\; y :: \mathit{Ent} \;.\; y - S\ O = P\ y$$

Ya que $(-)$ está definida por patrones en el segundo argumento, la propiedad (4.4) es inmediata:

$$\begin{array}{ll} & y - S\ O = P\ y \\ \equiv & \{\ 2)-\ \} \\ & P(y - O) = P\ y \end{array}$$

$$
\begin{array}{ll}
\equiv & \{\, 1)- \,\} \\
& P\ y\ =\ P\ y \\
\equiv & \\
& \mathit{Cierto}
\end{array}
$$

Sin embargo, la propiedad (4.3) es difícil de establecer por inducción. El caso base es trivial. El primer caso inductivo no ofrece problema:

$$
\begin{array}{ll}
& S\ y\ +\ S\ O\ =\ S\ (S\ y) \\
\equiv & \{\, 2)+ \,\} \\
& S(y + S\ O)\ =\ S\ (S\ y) \\
\equiv & \{\ \text{hipótesis de inducción}\ \} \\
& S(S\ y)\ =\ S(S\ y) \\
\equiv & \\
& \mathit{Cierto}
\end{array}
$$

Pero el segundo paso inductivo conduce a una condición adicional:

$$
\begin{array}{ll}
& P\ y\ +\ S\ O\ =\ S\ (P\ y) \\
\equiv & \{\, 3)+ \,\} \\
& P(y + S\ O)\ =\ S\ (P\ y) \\
\equiv & \{\ \text{hipótesis de inducción}\ \} \\
& P(S\ y)\ =\ S(P\ y)
\end{array}
$$

Tal condición no puede probarse y debe establecerse como un axioma.

En definitiva, la demostrabilidad de la propiedad del apartado (D) es posible solamente para ciertas definiciones del producto. Algo similar ocurre con la conmutatividad de la suma.

Solución al Ejercicio 18.62 (pág. 548).– **(A)**.– Basta considerar las siguientes funciones:

$$
\begin{array}{l}
\mathit{éxitos}\ os\ =\ \mathit{length}\ (\mathit{resuelve}\ os) \\
\mathit{resuelve}\ ::\ \mathit{Secuencia} \rightarrow [\mathit{Resultado}] \\
\mathit{resuelve}\ [\,]\ \ \ \ \ \ \ \ =\ [\mathit{Éxito}] \\
\mathit{resuelve}\ (o : os)\ =\ \mathit{concat}\ [\ \mathit{resuelve}\ (cs +\!\!+\ os)\ | \\
\qquad\qquad\qquad\qquad o'\ :-\ cs\ \leftarrow\ \mathit{miPrograma},\ o' == o\ ]
\end{array}
$$

Observa que la lista por comprensión captura el motor de resolución de PROLOG: tomar una cláusula cuya cabeza unifique con el primer objetivo de la secuencia, e intentar resolver la lista de objetivos que resulta de añadir al cuerpo (cs) el resto de objetivos (os).

(B).– En la función $\mathit{resuelve}2$ el segundo argumento es un acumulador (la traza recorrida hasta ese momento):

$$
\begin{array}{l}
\mathit{resuelve}2\ [\,]\ \ \ \ \ \ t\ =\ [t] \\
\mathit{resuelve}2\ (o : os)\ t\ =\ \mathit{concat}\ [\ \mathit{resuelve}2\ (cs +\!\!+\ os)\ (t +\!\!+\ [o])\ | \\
\qquad\qquad\qquad\qquad o'\ :-\ cs\ \leftarrow\ \mathit{miPrograma},\ o' == o\ ] \\
\mathit{trazar}\ os\ =\ \mathit{resuelve}2\ os\ [\,]
\end{array}
$$

Como vemos, al elegir una cláusula válida ($o' :- cs \leftarrow miPrograma,\ o' == o$) tratamos de resolver la misma lista de objetivos que en el caso anterior ($cs +\!\!+ os$) pero la nueva traza hasta ese momento pasa a ser $t +\!\!+ [o]$. Si finalmente llegamos a un éxito ($resuelve\ [\,]\ t$) devolvemos en forma de lista ($[t]$) la traza alcanzada (es importante devolverla en forma de lista ya que en otro caso se mezclarían las distintas trazas).

Solución al Ejercicio 18.63 (pág. 550).– **(A).**– Para calcular el retraso máximo basta con recorrer recursivamente el circuito

```
maxRetraso                    :: Circuito → Retraso
maxRetraso (AND c1 c2 r)      = r + max (maxRetraso c1) (maxRetraso c2)
maxRetraso (OR c1 c2 r)       = r + max (maxRetraso c1) (maxRetraso c2)
maxRetraso (NAND c1 c2 r)     = r + max (maxRetraso c1) (maxRetraso c2)
maxRetraso (NOT c r)          = r + maxRetraso c
maxRetraso (BIT _)            = 0
maxRetraso (VAL _)            = 0
```

El siguiente operador calcula la unión de dos conjuntos:

```
-- Unión de conjuntos
(\/)                  :: Eq a ⇒ [a] → [a] → [a]
[] \/ ys              = ys
(x : xs) \/ ys
  | x `elem` ys       = xs \/ ys
  | otherwise         = x : xs \/ ys
```

A partir de éste es fácil definir la función *entradas*:

```
entradas                    :: Circuito → [Var]
entradas (AND c1 c2 _)      = entradas c1 \/ entradas c2
entradas (OR c1 c2 _)       = entradas c1 \/ entradas c2
entradas (NAND c1 c2 _)     = entradas c1 \/ entradas c2
entradas (NOT c _)          = entradas c
entradas (BIT _)            = []
entradas (VAL a)            = [a]
```

(B).– El operador $(>+<)$ realiza la unión de de dos listas con nombres de puertas y sus contadores asociados. A partir de éste definimos la función *puertas*:

```
type NombreC = String

puertas                     :: Circuito → [(NombreC, Int)]
puertas (AND c1 c2 _)       = [("and", 1)] >+< puertas c1 >+< puertas c2
puertas (OR c1 c2 _)        = [("or", 1)] >+< puertas c1 >+< puertas c2
puertas (NAND c1 c2 _)      = [("nand", 1)] >+< puertas c1 >+< puertas c2
puertas (NOT c _)           = [("not", 1)] >+< puertas c
puertas (BIT _)             = []
puertas (VAL _)             = []
```

```
(>+<) :: [(NombreC, Int)] → [(NombreC, Int)] → [(NombreC, Int)]
(>+<) = foldr insertar
  where
    insertar x []                    = [x]
    insertar p@(c, n) ((c', n') : cs)
      | c == c'                      = (c, n + n') : cs
      | otherwise                    = (c', n') : insertar p cs
```

La función *entradas* usando el plegado es:

```
entradas' :: Circuito → [Var]
entradas' = foldC f f f (λ es _ → es) (const []) (: [])
  where
    f x y _ = x \/ y
```

(C).– Definimos primero funciones para representar las distintas operaciones booleanas:

```
toBool   :: Bit → Bool
toBool 0 = False
toBool 1 = True

fromBool       :: Bool → Bit
fromBool False = 0
fromBool True  = 1

a &&& b  = fromBool (toBool a && toBool b)
a ||| b  = fromBool (toBool a || toBool b)
no       = fromBool . not . toBool
a &|& b  = no (a &&& b)   -- NAND
```

Y a partir de éstas:

```
eval                       :: Circuito → [Var] → Bit
eval (AND c1 c2 _) xs      = eval c1 xs &&& eval c2 xs
eval (OR c1 c2 _) xs       = eval c1 xs ||| eval c2 xs
eval (NAND c1 c2 _) xs     = eval c1 xs &|& eval c2 xs
eval (NOT c _) xs          = no (eval c xs)
eval (BIT b) xs            = b
eval (VAL a) xs            = fromBool (a `elem` xs)
```

Para la tabla de verdad necesitaremos la función que calcule las partes de un conjunto:

```
partes          :: [a] → [[a]]
partes []       = [[]]
partes (x : xs) = partesXs ++ map (x :) partesXs
  where
    partesXs = partes xs
```

Y evaluamos el circuito con todas las posibles entradas:

```
tablaVerdad  :: Circuito → [([Var], Bit)]
tablaVerdad c = zip combs (map (eval c) combs)
  where
    ents  = entradas c
    combs = partes ents
```

(D).– Se trata de convertir cada puerta en su implementación con puertas AND y NOT:

```
andYNot :: Circuito → Circuito
andYNot = foldC (λ c1 c2 _ → pAnd c1 c2)
                (λ c1 c2 _ → pNot (pAnd (pNot c1) (pNot c2)))
                (λ c1 c2 _ → pNot (pAnd c1 c2))
                (λ c1 _ → pNot c1)
                BIT
                VAL
  where
    pAnd x y = AND x y 3
    pNot x   = NOT x 1
```

Para usar solo puertas NAND podemos obtener primero un circuito con tan solo puertas AND y NOT e implementar estas últimas con puertas NAND:

```
soloNand :: Circuito → Circuito
soloNand = soloNand' . andYNot
  where
    soloNand' = foldC (λ c1 c2 _ → let c3 = pNAnd c1 c2 in pNAnd c3 c3)
                      (error "imposible")
                      (error "imposible")
                      (λ c1 _ → pNAnd c1 c1)
                      BIT
                      VAL
    pNAnd x y = NAND x y 5
```

19.5. AÑO 2000

SOLUCIÓN A 18.64 (PÁG. 554)
(A).–

```
deCola (c :> (U x)) = (c, x)
deCola (c :> c')    = (c :> c'', x) where (c'', x) = deCola c'
```

(B).– `redCola :: (b → b → b) → (a → b) → Cola a → b`
(C).– La función *cosa* calcula el número de elementos de una cola, ya que la función anterior verifica las ecuaciones:

$$
\begin{array}{ll}
(1) & cosa\ (U\ x\ ) \quad = 1 \\
(2) & cosa\ (c\ :>\ c') \ = \ cosa\ c\ +\ cosa\ c'
\end{array}
$$

La primera es trivial y la segunda sigue directamente de la definición.
(D).– Procedemos por inducción sobre la cola c
Caso Base ($c \equiv U\ x$). Basta aplicar la ecuación (1) de $cosa$.
Paso Inductivo. Hay que probar

$$\forall c, c' \centerdot c, c' :: Cola\ a \centerdot 0 < cosa\ c \wedge 0 < cosa\ c' \ \Rightarrow\ 0 < cosa(c :> c')$$

y para ello basta aplicar la ecuación (2) de $cosa$.
(E).– El tipo pedido es

$$curiosa\ ::\ (a\ \rightarrow\ b)\ \rightarrow\ Cola\ a\ \rightarrow\ Cola\ b$$

y $curiosa\ h$ aplica la función h a todos los elementos de una cola, ya que tal función verifica las ecuaciones

$$
\begin{array}{ll}
(1) & curiosa\ h\ (U\ x\ ) \quad = \ U\ (h\ x) \\
(2) & curiosa\ h\ (c :> c') \ = \ curiosa\ h\ c\ :>\ curiosa\ h\ c'
\end{array}
$$

(F).– $concaCola\ =\ redCola\ (:>)\ id$

Solución al Ejercicio 18.65 (pág. 554).–
(A).– $ss\ =\ 1\ :\ 2\ :\ zipWith3\ (\lambda\ n\ x\ x'\ =\ n * x' + x)\ [0..]\ ss\ (tail\ ss)$
(B).–

$$
\begin{array}{l}
sol\ =\ calcula\ ss\ [0..] \\
calcula\ (x : x' : x'' : xs)\ (n : nn) \\
\quad |\ x''\ -\ n * x\ +\ x'\ >\ 1000\ =\ n \\
\quad |otherwise \qquad\qquad\qquad\quad =\ calcula\ (x' : x'' : xs)\ nn
\end{array}
$$

Solución al Ejercicio 18.66 (pág. 555).– **(A).**–

$$
\begin{array}{l}
fusion\ =\ (:>) \\
listaACola\ [x]\ =\ U\ x \\
listaACola\ xs\ =\ listaACola\ us\ :>\ listaACola\ us' \\
\quad \textbf{where}\ (us, us')\ =\ partir\ xs \\
\qquad\qquad partir\ xs\ =\ splitAt\ (length\ xs\ `div`\ 2)\ xs
\end{array}
$$

(B).– El tipo es $pamCola\ ::\ a\ \rightarrow\ Cola\ (a\ \rightarrow\ b)\ \rightarrow\ Cola\ b$ y la función aplica las funciones de una cola (de funciones) al primer argumento, devolviendo una cola de resultados:

$$
\begin{array}{l}
\text{MAIN>}\ \ pamCola\ 3\ (U\ (+2)\ :>\ U\ (*3)) \\
U\ 5\ :>\ U\ 9\ ::\ Cola\ Integer
\end{array}
$$

Otra función útil es la función que aplica una función a una cola de objetos:

$$
\begin{aligned}
&mapCola :: (a \to b) \to Cola\ a \to Cola\ b\\
&mapCola\ h = redCola\ (:>)\ (U\ .\ h)
\end{aligned}
$$

Otras funciones interesantes pueden ser la siguientes, cuyo significado viene dado por su identificador:

$$
\begin{aligned}
&colaAlista :: Cola\ a \to [a]\\
&colaAlista = redCola\ (++)\ (:[])
\end{aligned}
$$

$$
\begin{aligned}
&enCola :: a \to Cola\ a \to Cola\ a\\
&enCola\ z\ (U\ x) = U\ z :>\ U\ x\\
&enCola\ z\ (c :>\ c') = enCola\ z\ c :>\ c'
\end{aligned}
$$

$$
\begin{aligned}
&deCola :: Cola\ a \to (Cola\ a,\ a)\\
&deCola\ (c :>\ (U\ x)) = (c, x)\\
&deCola\ (c :>\ c') = (c :>\ c'', x)\ \mathbf{where}\ (c'', x) = deCola\ c'
\end{aligned}
$$

$$
\begin{aligned}
&concaCola :: Cola\ (Cola\ a) \to Cola\ a\\
&concaCola = redCola\ (:>)\ id
\end{aligned}
$$

(C).– Las funciones pedidas son

$$
\begin{aligned}
&sumaCola = redCola\ (+)\ id\\
&pertenece\ x = redCola\ (||)\ (==x)
\end{aligned}
$$

(D).–

$$
\begin{aligned}
&valorEn\ x = redCola\ (+)\ (\lambda\ (n, c) \to c * x\hat{}n)\\
&mulPorMonomio\ (n, c) = mapCola\ (\lambda\ (n', c') \to (n + n', c * c'))
\end{aligned}
$$

Otra funciones interesantes son (si cada grado aparece una sola vez):

$$
\begin{aligned}
&sumaMonomio :: Monomio \to Polinomio \to Polinomio\\
&sumaMonomio\ (n, c)\ (U\ (n', c'))\\
&\quad |\ n == n' = U(n, c + c')\\
&\quad |\ otherwise = U(n, c') :>\ U(n, c)\\
&sumaMonomio\ (n, c)\ (p :>\ p')\\
&\quad |\ estáGrado\ n\ p = sumaMonomio\ (n, c)\ p :>\ p'\\
&\quad |\ otherwise = p :>\ sumaMonomio\ (n, c)\ p'
\end{aligned}
$$

$$
\begin{aligned}
&estáGrado :: Integer \to Polinomio \to Bool\\
&estáGrado\ n = redCola\ (||)\ (\lambda\ (n', \_) \to n == n')
\end{aligned}
$$

Solución al Ejercicio 18.67 (pág. 556).–
(A).– El tipo es $recons :: (a \to b \to a) \to a \to [b] \to [a]$.
(B).– La Figura 19.5 captura el cómputo de la función.
(C).–

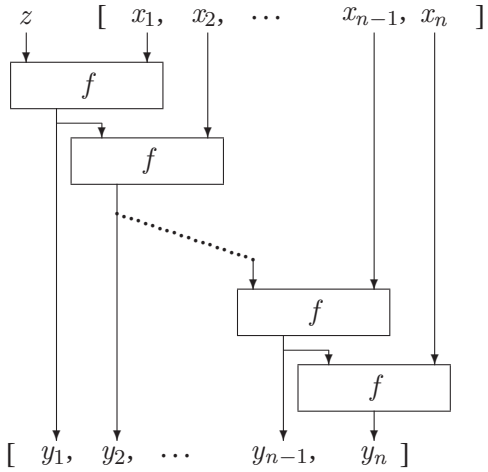

Figura 19.5: Resultado de la función $recons$..

$$
\begin{array}{l}
separa\ [x] \quad\ = ([\,], x) \\
separa\ (x : xs) \ = (x : ys, u)\ \mathbf{where}\ (ys, u) \ = \ separa\ xs
\end{array}
$$

(D).– Procedemos por inducción sobre la lista:
Caso Base ($xs \equiv [\,]$)

$$
\begin{array}{ll}
 & us +\!\!+ [y]\ \mathbf{where}\ (us, y) \ = \ separa\ [x] \\
\equiv & \{\text{ def. de separa }\} \\
 & us +\!\!+ [y]\ \mathbf{where}\ (us, y) \ = \ ([\,], x) \\
\equiv & \{\text{ elimamos el cualificador }\} \\
 & [\,] +\!\!+ [x] \\
\equiv & \{\text{ def. de } +\!\!+ \} \\
 & [x]
\end{array}
$$

Paso Inductivo

$$
\begin{array}{ll}
 & us +\!\!+ [y]\ \mathbf{where}\ (us, y) \ = \ separa\ (x : xs) \\
\equiv & \{\text{ def. de separa }\} \\
 & us +\!\!+ [y]\ \mathbf{where}\ \{(us, y) \ = \ (x : xs, u);\ (ys, u) \ = \ separa\ xs\} \\
\equiv & \{\text{ sustituyendo según cualificador }\} \\
 & (x : ys) +\!\!+ [u]\ \mathbf{where}\ (ys, u) \ = \ separa\ xs \\
\equiv & \{\text{ definición de } +\!\!+ \} \\
 & x : (ys +\!\!+ [u]\ \mathbf{where}\ (ys, u) \ = \ separa\ xs) \\
\equiv & \{\ \text{hipótesis de inducción }\} \\
 & x : xs
\end{array}
$$

(E).–

$$sumar = snd\ .\ separa\ .\ recons\ (+)\ 0$$
$$máximo\ (x : xs) = (snd\ .\ separa\ .\ recons\ max\ x)\ xs$$

(F).– $ruffini\ u = separa\ .\ recons\ (\ (+)\ .\ (*u)\ )\ 0$

Solución al Ejercicio 18.68 (pág. 556).– **(A)**.– $pam'\ x = map\ (\$\ x)$
(B).– *Caso Base* (trivial)
Paso Inductivo:

$$\begin{array}{ll} & pam\ x\ (f : fs) = pam'\ x\ (f : fs) \\ \equiv & \{\text{ def. }\} \\ & f\ x\ :\ pam\ x\ fs = map\ (\$\ x)\ (f : fs) \\ \equiv & \{\text{ h.i., def. de } map \text{ y } (\$x) = f\ x\ \} \\ & f\ x\ :\ pam'\ x\ fs = f\ x\ :\ map\ (\$\ x)\ fs \\ \equiv & \{\text{ def. de } pam'\ \} \\ & Cierto \end{array}$$

Solución al Ejercicio 18.69 (pág. 556).– **(A)**.– Una posible definición para las funciones son:

$$\begin{array}{l} seJugó :: PartidoJugado \to Torneo \to Bool \\ seJugó\ (x, y, \text{"1"})\ t = elem\ (x, y, \text{"1"})\ t \mid\mid elem\ (y, x, \text{"2"})\ t \\ seJugó\ (x, y, \text{"2"})\ t = elem\ (x, y, \text{"2"})\ t \mid\mid elem\ (y, x, \text{"1"})\ t \\ seJugó\ (x, y, \text{"x"})\ t = elem\ (x, y, \text{"x"})\ t \mid\mid elem\ (y, x, \text{"x"})\ t \end{array}$$

$$\begin{array}{l} seJugaron :: [PartidoJugado] \to Torneo \to Bool \\ seJugaron\ [\,]\ \ \_ = True \\ seJugaron\ (p : ps)\ t = seJugó\ p\ t\ seJugaron\ ps\ t \end{array}$$

que de una manera más simple podría haberse escrito también como:

$$seJugaron\ ps\ t = foldr\ (\lambda\ p\ b \to b\ seJugó\ p\ t)\ True\ ps$$

(B).– Teniendo en cuenta que la función $paises$ está definida como

$$\begin{array}{l} países :: [Equipo] \\ países = [\text{"esp"}, \text{"nor"}, \text{"yug"}, \text{"esl"}] \end{array}$$

Entonces

$$\begin{array}{l} enfrentamientos :: [Partido] \\ enfrentamientos = [(x, y) \mid x \leftarrow países,\ y \leftarrow países, x < y] \end{array}$$

donde la relación $x < y$ nos asegura que ni repetimos equipo ni consideramos los simétricos.

Por otro lado, tenemos

```
juegaTorneo :: [Partido] → [Resultado] → Torneo
juegaTorneo ps rs = zipWith (λ(x, y) r → (x, y, r)) ps rs
```

(C).– Recordemos la función *columnas*

```
columnas :: Int → [[Resultado]]
columnas 0       = [[]]
columnas (n + 1) = map ("1" :) cs ++ map ("X" :) cs ++ map ("2" :) cs
                   where cs = columnas n
```

Entonces

```
resultados :: [Torneo]
resultados = map (juegaTorneo enfrentamientos)
                 (columnas (length enfrentamientos))
```

y

```
puntos :: [PartidoJugado] → [Puntos]
puntos []       = []
puntos (p : ps) = acumula p (puntos ps)
```

o lo que es lo mismo

```
puntos = foldr acumula
```

donde la función *acumula* viene definida por:

```
acumula :: PartidoJugado → [Puntos] → [Puntos]
acumula (x, y, "1") p = incrementa x 3 p
acumula (x, y, "X") p = incrementa x 1 (incrementa y 1 p)
acumula (x, y, "2") p = incrementa y 3 p

incrementa :: Equipo → Int → [Puntos] → [Puntos]
incrementa x n [] = [(x, n)]
incrementa x n ((y, m) : xs)
  | x == y    = (x, n + m) : xs
  | otherwise = (y, m) : incrementa x n xs
```

(D).–

```
ganadores :: [Puntos] → ([Equipo], Int)
ganadores [(p, v)] = ([p], v)
ganadores ((p, v) : xs)
  | v > v'  = ([p], v)
  | v == v' = (p : ps, v)
  | v < v'  = (ps, v')
    where (ps, v') = ganadores xs
```

```
posibilidad :: Equipo → [PartidoJugado] → [Torneo]
posibilidad e ps = [t | (t, p) ← tsp,
                        let (ps, n) = ganadores p,
                        elem e ps]
  where
    ts  = filter (seJugaron ps) resultados
    tsp = zip ts (map puntos ts)
```

Solución al Ejercicio 18.70 (pág. 559).–
(A).– Una demostración parecida para *fmap* puede verse en 11.2.1.
(B).– La funciones pedidas son:

```
frontera :: Htree a → [a]
frontera (H x)     = [x]
frontera (i :↑: d) = frontera i ++ frontera d

revHtree :: Htree a → Htree a
revHtree (H x)     = H x
revHtree (i :↑: d) = revHTree d :↑: revHTree i
```

y el plegado para esta estructura es:

```
foldHtree :: (b → b → b) → (a → b) → Htree a → b
foldHtree f g (H x)     = g x
foldHtree f g (i :↑: d) = f (foldHtree f g i) (foldHtree f g d)
```

con lo que unas nuevas definiciones para las funciones anteriores basadas en este plegado serán:

```
frontera = foldHtree (++) (:[])
revHtree = foldHtree (flip (:↑:)) H
```

donde vemos que utilizamos el constructor H como una función.

Por otro lado, la función *anivel* se define como:

```
anivel :: Htree a → Htree Int
anivel ar = anivel' ar 0

anivel' :: Htree a → Int → Int
anivel' (H x)     n = H n
anivel' (i :↑: d) n = anivel' i (n + 1) :↑: anivel' d (n + 1)
```

(C).– La función *sumas* puede fácilmente escribirse como

```
sumas :: [Int]
sumas = 1 : zipWith (+) [2..] sumas
```

y representarse en una red de procesos como la de la Figura 19.6.

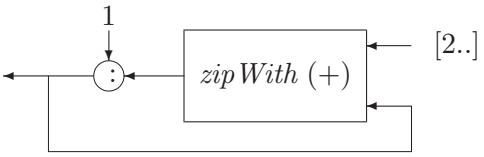

Figura 19.6: Red que representa a la función $suma$..

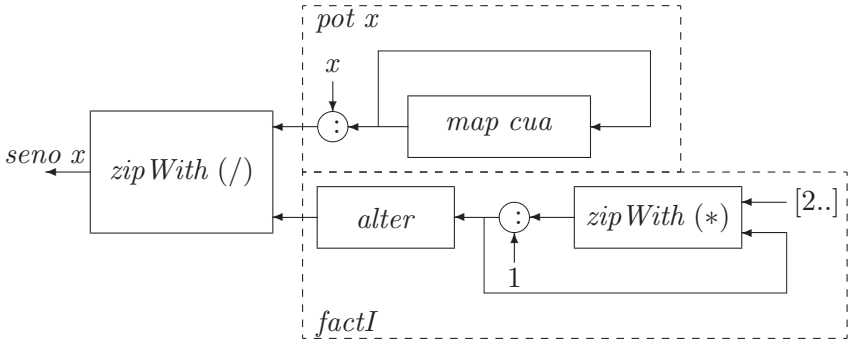

Figura 19.7: Red que representa a la función $seno\ x$..

Solución al Ejercicio 18.72 (pág. 560).– **(A)**.–

$$nodos :: ArbGen\ a \rightarrow Int$$
$$nodos = foldAG\ (\lambda\, x\ xs \rightarrow 1 + sum\ xs)$$

(B).–

$$preorden :: ArbGen\ a \rightarrow [a]$$
$$preorden = foldAG\ (\lambda\, x\ xs \rightarrow x : concat\ xs)$$

Solución al Ejercicio 18.73 (pág. 560).–

$$pot\ x = x : map\ cua\ (pot\ x)\ \mathbf{where}\ cua\ y = -y * x * x$$
$$factI = alter\ fac\ \mathbf{where}\ fac = 1 : zipWith\ (*)\ fac\ [2..]$$
$$alter\ (x : x' : xs) = x : alter\ xs$$
$$coef\ x = zipWith\ (/)\ (pot\ x)\ factI$$

La correspondiente red podría ser la- de la Figura 19.7.

Solución al Ejercicio 18.74 (pág. 560).–

(A).–

```
votos :: PartidoYVotos → NumeroDeVotos
votos (P n x) = x

insertaV :: PartidoYVotos → [PartidoYVotos] → [PartidoYVotos]
insertaV v [] = [v]
insertaV v (v' : vs)
  | votos v > votos v' = v : v' : vs
  | otherwise          = v' : insertaV v vs
```

(B).– $ordenaV\ vs = foldr\ insertaV\ [\,]\ vs$

(C).–

```
sumaV v [] = [v]
sumaV (P n x) (P n' y : vs)
  | n == n'   = P n (x + y) : vs
  | otherwise = insertaV (P n' y) (sumaV (P n x) vs)
```

(D).– $juntarV = foldr\ sumaV$

(E).–

```
d'Hont 0 vs                  = []
d'Hont n (P nom vot : xs) =
  sumaV (P nom 1)
        (d'Hont (n − 1) (insertaV (P nom (vot `div` 2)) xs))
```