1 Funciones conocidas

```
foldr :: (a -> b -> b) -> b -> [a] -> b
foldr f z [] = z
foldr f z (x:xs) = f x (foldr f z xs)
```

```
foldl ... (tambien version con foldr)
```

```
recr :: (a -> [a] -> b -> b) -> b -> [a] -> b
recr f z [] = z
recr f z (x:xs) = f x xs (recr f z xs)
```

```
reverse :: [a] -> [a] -> [a]
reverse [] = []
reverse (x:xs) = reverse xs ++ [xs]
-- Def. alternativa reverse
reverse = foldr (\\x rec -> rec ++ (x:[])) []
```

```
map :: (a -> b) -> [a] -> [b]
map f [] = []
map f (x:xs) = f x : (map f xs)
```

```
... version de map con foldr
```

2 Esquemas de recursión

2.1 Recursión estructural

Sea $g :: [a] \to b$

$$g\ [\,] = \langle CB \rangle(z) \tag{1}$$
$$g\ (x : xs) = \langle CR \rangle(...x\ ...\ (g\ xs)...) \tag{2}$$

Los casos base CB_i devuelven z_i que no dependen de g, mientras que los casos recursivos no tienen las variables g ni xs por separado, excepto en la expresión $(g\ xs)$. Podría aparecer de la forma g ... xs ... en caso que g tomara parámetros en cierto orden, pero nunca de la forma $g\ (f\ xs)$ (es decir, operar sobre el contenido de xs antes de llamar a la recursión) ya que esto no garantiza que la lista es cada vez mas pequeña.

Foldr abstrae el esquema de recursión estructural, y decimos que toda recursión estructural es una instancia de foldr.

2.2 Recursión primitiva

Sea $g :: [a] \to b$

$$g\ [\,] = \langle CB \rangle(z) \tag{3}$$
$$g\ (x : xs) = \langle CR \rangle(...x\ ...\ \boldsymbol{xs}\ ...\ (g\ xs)...) \tag{4}$$

Análogo al esquema de recursión estructural, pero permite referirse a xs fuera del término $(g\ xs)$.

Toda recursión primitiva es una instancia de **recr**.

2.3 Recursión iterativa

Sea $g :: b \to [a] \to b$

$$g\ ac\ [\,] = \langle CB \rangle (z) \tag{5}$$
$$g\ ac\ (x : xs) = \langle CR \rangle (...x\ ...\ xs\ ...\ \boldsymbol{(g\ ac'\ xs)}...) \tag{6}$$

Invoca a $(g\ ac'\ xs)$ donde $ac' = f\ (...ac\ ...x)$ es el nuevo acumulador actualizado en función de su valor anterior y la variable x.

Toda recursión iterativa es una instancia de **foldl**.

$$foldr\ (\boxplus)\ z\ [a, b, c] = a\ \boxplus\ (b \boxplus\ (c \boxplus\ z))$$
$$foldl\ (\boxplus)\ z\ [a, b, c] = ((z \boxplus\ a) \boxplus\ b) \boxplus\ c$$

3 Tipos de datos algebráicos

3.1 Tipos enumerados

Constructores de tipo no extensibles, y declara que son los únicos constructores de ese tipo.

```
data Tipo = Const1 | Const2 | Const3 | ...
```

3.2 Tipos producto

```
data Tipo = Constuctor T1 T2 T3
```

Donde T1 T2 y T3 son tipos, es decir un constructor con muchos parametros.

3.3 Tipos recursivos

Son tipos de datos inductivos, tiene un constructor base "CBase" y un constructor que toma al menos un parametro del tipo que lo está definiendo.

```
data Tipo = CBase | Constructor Tipo
```

3.4 Caso general

Los tipos algebráicos tienen la forma

$$\begin{aligned}
\text{data } T = {} & CB_1\langle cb_params_1 \text{ sin el tipo } T\rangle \\
\mid \quad & \vdots \\
\mid \quad & CB_1\langle cb_params_n \text{ sin el tipo } T\ \rangle \\
\mid \quad & CR_1\langle cr_params_1 \ldots\ T\rangle \\
\mid \quad & \vdots \\
\mid \quad & CR_1\langle cr_params_m \ldots\ T\rangle
\end{aligned}$$

3.5 Recursion estructural en general

Sea $g :: T \to \mathbb{Y}$

$$\begin{aligned}
g\ (CB_1\ \langle cb_params_1\rangle)\ \ldots &= \langle CB_1\rangle(z_1) \\
&\vdots \\
g\ (CB_n\ \langle cb_params_n\rangle)\ \ldots &= \langle CB_n\rangle(z_n) \\
g\ (CR_2\ \langle cr_params_1\rangle)\ \ldots &= \langle CR_1\rangle(z_1) \\
&\vdots \\
g\ (CR_m\ \langle cr_params_m\rangle)\ \ldots &= \langle CR_m\rangle(z_m)
\end{aligned}$$

Donde CB_k, CR_k es el k-iésimo caso base/recursivo, y cb_params_k, cr_params_k los parametros para ese constructor respectivamente.

Así como definiamos el fold en listas de tal forma que el tipo es

```
foldr :: (a -> b -> b) -> b -> [a] -> b
```

Esto significa que `(a -> b -> b)` es la función que va a manejar el constructor cons de listas `Cons List a`, y el `b` que aparece solo le corresponde al caso base de la lista vacía [ ].

De esta podemos definir la función fold para cualquier tipo recursivo armando una función por cada uno de sus constructores.

[TODO: falta poner mas info acá o un ejemplo]

4 Inducción estructural

4.1 Principio de inducción estructural

Sea $\mathcal{P}(y)$ un predicado sobre un $y :: \mathcal{T}$

$$\frac{\begin{array}{l}\forall cb\_params_1\ .\mathcal{P}(CB_1)\ \wedge \cdots \wedge\ \forall cb\_params_n\ .\ \mathcal{P}(CB_n) \\ \forall cr\_params_1\ .\forall x :: \mathcal{T}\ .(\mathcal{P}(x)\implies \mathcal{P}(CR_1 ... x ....)) \\ \quad\vdots \qquad\qquad\qquad \vdots \\ \forall cr\_params_m\ .\forall x :: \mathcal{T}\ .(\mathcal{P}(x)\implies \mathcal{P}(CR_m ... x ....))\end{array}}{\vdash \forall x :: \mathcal{T}\ .\ \mathcal{P}(x)}$$

5 Lemas de generación

Todos los tipos inductivos tienen un lema de generación, viene de la noción de como se construyen los tipos, si tenemos alguna expresión

$$...E... = ...$$

y sabemos que $E :: \mathcal{T}$ donde data $\mathcal{T} = A \mid B$ entonces sabemos que E solo puede haber sido construído como A o B.

En general tenemos que para algún tipo inductivo cualquiera

```
data Ttype a b = Nil | Const1 a | Const2 b | Const3 a b
```

Tenemos que algúna expresión E :: Ttype, E solo puede haber sido construido con uno de esos constructores, formalmente

si e :: Ttype entonces

$\exists x :: a\ .\exists y :: b\ .(E = \text{Nil})\ \vee\ (E = \text{Const1 x}) \vee\ (E = \text{Const2 y}) \vee\ (E = \text{Const3 x y})$

6 Principio de extensionalidad funcional

Desde un punto de vista **intensional** dos valores son iguales si están construidos de la misma manera. Desde el punto de vista **extensional** dos valores son iguales so son indistinguibles al observador.

Normalmente pensamos que si $f = g$ entonces $\forall x :: a\ .f\ x = g\ x$ pensandolo de forma inversa hariamos el principio de extensionalidad funcional que es la siguiente implicación:

si $\forall x :: a\ .f\ x = g\ x$ entonces $f = g$

[TODO: falta isomorfismo de tipos] [TODO: falta ejemplos de generalizacion de predicados de inductivos] [TODO: Lemas auxiliares]

7 Sistemas deductivos