Trabajo Práctico

Programación Funcional

2 de Abril de 2025 Paradigmas de Lenguajes de Programación

Integrante	LU	Correo electrónico
Solana Navarro	906/22	solanan3@gmail.com
Melli, Tomás Felipe	371/22	tomas.melli1@gmail.com
Lourdes Wittmund Montero	1103/22	lourdesmonterochiara@gmail.com
Marco Romano Fina	1712/21	marcoromanofinaa@gmail.com

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2610 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (++54 +11) 4576-3300
http://www.exactas.uba.ar

Índice

1 Introducción

Para este trabajo práctico vamos a implementar dos módulos.

1. **Documento** : donde definiremos un tipo de dato Doc y funciones para trabajar con docu- mentos. Un documento es una estructura cuyo objetivo es mostrarse por pantalla en forma amigable para el usuario.
2. **PPON** : donde definiremos un tipo de dato PPON y funciones para trabajar con este formato de datos compuesto. Además, definiremos cómo convertir un PPON a un Doc para mostrarlo en forma amigable.

2 Módulo Documento

Un documento está representado por la estructura recursiva Doc con la siguiente definición:

```
data Doc = Vacio | Texto String Doc | Linea Int Doc
```

- Vacio: representa un documento vacıo.
- Texto: representa un documento que contiene como primer componente un texto.
- Linea: representa un documento que contiene como primer componente un salto de linea. La siguiente linea comienza con una cantidad de espacios indicada por un entero.

Un valor del tipo Doc debe cumplir con los siguientes invariantes: Sea Texto s d entonces:

- s no debe ser el string vacıo.
- s no debe contener saltos de lınea
- d debe ser Vacio o Linea i d'

Sea Linea i d entonces:

- $i \geq 0$

Ya tenemos definidas las funciones : *vacío, linea y texto.*

2.1 Ejercicio 2

Usamos **foldDoc** para implementar el operador:

```
(<+>) :: Doc -> Doc -> Doc
d1 <+> d2 = foldDoc
        d2
        (\s rec -> if primerComandotext rec
            then Texto (s++(consigoString rec)) (consigoproxDoc rec)
            else Texto s rec)
        (\i rec -> Linea i rec)
        d1
```

2.1.1 Justificación de validez del invariante

- Sea Texto s d :
    1. Sea d1 $<+>$ d2, sabiendo que d1 y d2 cumplen con el invariante, por tanto, no tiene un string vacío, por tanto, en la unión de ambos no contendrán vacío ya que concatenamos dos cadenas de texto que no son vacíos (string no vacíos).
    2. Sea d1 $<+>$ d2, sabiendo que d1 y d2 cumplen con el invariante, por tanto, no tiene un salto de línea, la unión de ambos implica concatenar dos String que ya cumplen el invariante.
    3. Sea d1 $<+>$ d2, que cumplen el invariante, al concatenarlos, el único momento donde podemos encontrarnos frente a una situación conflictiva es al momento de concatenar un documento *d1* finalizado en una forma Texto s Vacio y donde *d2* comience con Texto s d. Para dar solución y mantener el invariante, nos aseguramos de concatenarlos de forma tal que no quede un documento del formato Texto s d donde d es Texto s´ d´.
- Sea Linea i d : dados d1,d2 que cumplen con el invariante, en la implementación utilizamos el constructor (Linea i d), por tanto queda tal cual como estaba.

2.2 Ejercicio 3

```
indentar :: Int -> Doc -> Doc
indentar i d = foldDoc
              vacio
              (\txt rec -> Texto txt rec)
              (\espacios rec -> Linea (espacios + i) rec) d
```

2.2.1 Justificación de validez del invariante

- Sea Texto s d :
  1. No se modifica, por tanto, no contendrá un s vacío
  2. No se modifica, por tanto, no contendrá un s de salto de línea.
  3. No se modifica, por tanto, se mantiene el invariante.
- Sea Linea i´ d : como el documento que indentamos cumple el invariante y por precondición donde i es parámetro de indentar $i > 0$, entonces, se cumple que $i \geq 0$.

3 Módulo PPON

```
data PPON = TextoPP String | IntPP Int | ObjetoPP [(String, PPON)] deriving (Eq, Show)
```

3.1 Ejercicio 9

```
pponADoc :: PPON -> Doc
pponADoc (TextoPP s) = texto (show s)
pponADoc (IntPP i) = texto (show i)
pponADoc (ObjetoPP pares) =
  if pponObjetoSimple (ObjetoPP pares)
    then aplanar (entreLlaves (map (\(x,y) -> texto (show x) <+> texto ": " <+> pponADoc y) pares))
    else entreLlaves (map (\(x,y) -> texto (show x) <+> texto ": " <+> pponADoc y) pares)
```

3.1.1 Justificación del tipo de recursión

El tipo de recursión es **estructural.** Esto se debe a que la función realiza la recursión sobre la estructura de la segunda coordenada de la tupla en el contexto del *map* dentro del if. Además, podemos confirmar que no se trata de una recursión *primitiva* dado que el y sólo es utilizado en la llamada recursiva y no es se hace uso de este dato fuera del contexto de ese llamado.
Por otra parte, la recursión utilizada es *explícita* ya que es visible el llamado de la función a sí misma en el código.

4 Demostración

4.1 Ejercicio 10

Demostrar, utilizando razonamiento ecuacional e inducción estructural, que para todo n, m :: `Int` positivos y x :: `Doc` se cumple:

$$\texttt{indentar}\ n(\texttt{indentar}\ m\ x) = \texttt{indentar}\ (n+m)\ x$$

Se sugiere demostrar los siguientes lemas :

1. $indentar\ k\ Vacio = Vacio\ \forall k :: Int\ positivo$
2. $indentar\ k\ (Texto\ s\ d) = Texto\ s\ (indentar\ k\ d)\ \forall k :: Int\ positivo, s :: String\ y\ d ::: Doc$
3. $indentar\ k\ (Linea\ k\ d) = Linea\ (m+k)\ (indentar\ m\ d))\ \forall m, k :: Int\ positivo\ y\ d ::: Doc$

Tenemos las siguientes funciones con sus etiquetas para realizar la correcta demostración de los lemas y, consecuentemente, el ejercicio.

$vacio :: Doc$
$vacio = Vacio\ \{V_0\}$

$linea :: Doc$
$linea = Linea\ 0\ Vacio\ \{L_0\}$

$texto :: String -> Doc$
$textot|''`elem`t = error''''\{T_1\}$
$texto\ [\,] = Vacio\ \{T_2\}$
$texto\ t = Texto\ t\ Vacio\ \{T_3\}$

$foldDoc :: b -> (String -> b -> b) -> (Int -> b -> b) -> Doc -> b$
$foldDoc\ fVacio\ fTexto\ fLinea\ doc = case\ doc\ of$
$Vacio -> fVacio\ \{FD_1\}$
$Texto\ s\ d -> fTexto\ s\ (rec\ d)\ \{FD_2\}$
$Linea\ i\ d -> fLinea\ i\ (rec\ d)\ \{FD_3\}$
$where\ rec = foldDoc\ fVacio\ fTexto\ fLinea\ \{FD_4\}$

$indentar :: Int -> Doc -> Doc$
$indentar\ i = foldDoc\ \ vacio\ \ (text\ rec -> Texto\ text\ rec)\ \ (line\ rec -> Linea\ (line + i)\ rec)\ \{I_0\}$

4.1.1 Probamos Lema 1

$$indentar\ k\ Vacio = Vacio\ \forall k :: Int\ positivo$$

$\overset{ppio.\ de\ reemplazo}{=} foldDoc\ vacio\ (text\ rec -> Texto\ text\ rec)\ (line\ rec -> Linea\ (line+k)\ rec))\ Vacio = Vacio$
$\overset{FD_1}{=} vacio = Vacio$
$\overset{V_0}{=} Vacio = Vacio$
Como se quería probar.

4.1.2 Probamos Lema 2

$$indentar\ k\ (Texto\ s\ d) = Texto\ s\ (indentar\ k\ d)\ \forall k :: Int\ positivo, s :: String\ y\ d ::: Doc$$

$\overset{ppio\ de\ reemplazo}{=} foldDoc\ vacio\ (text\ rec -> Texto\ text\ rec)\ (line\ rec -> Linea\ (line+k)\ rec))\ (Texto\ s\ d)$
$\overset{FD_2}{=} (text\ rec -> Texto\ text\ rec)\ s\ (foldDoc\ vacio\ (text\ rec -> Texto\ text\ rec)\ (line\ rec -> Linea\ (line+k)\ rec)\ d)$
$\overset{\beta\ y\ ppio.\ de\ reemplazo}{=} (rec -> Texto\ s\ rec)\ (indentar\ k\ d)$
$\overset{\beta}{=} Texto\ s\ (indentar\ k\ d)$
Como se quería probar.

4.1.3 Probamos Lema 3

$$indentar\ m\ (Linea\ k\ d) = Linea\ (m+k)\ (indentar\ m\ d))\ \forall m, k :: Int\ positivo\ y\ d ::: Doc$$

$\overset{ppio.\ de\ reemplazo}{=} foldDoc\ vacio\ (text\ rec->Texto\ text\ rec)\ (line\ rec->Linea\ (line+m)\ rec)\ (Linea\ k\ d)$
$\overset{FD_3}{=} (line\ rec->Linea\ (line+m)\ rec)\ k\ (foldDoc\ vacio\ (text\ rec->Texto\ text\ rec)\ (line\ rec->Linea\ (line+m)\ rec)\ d)$
$\overset{ppio.\ de\ reemplazo}{=} (line\ rec->Linea\ (line+m))\ k\ (indentar\ m\ d)$
$\overset{\beta}{=} (rec->Linea\ (k+m)\ rec)\ (indentar\ m\ d)$
$\overset{\beta}{=} Linea\ (k+m)\ (indentar\ m\ d)$
$\overset{conmutatividad\ de\ la\ suma}{=} Linea\ (m+k)\ (indentar\ m\ d)$
Como se quería probar.

4.1.4 Demostración Ejercicio

$$\texttt{indentar}\ n(\texttt{indentar}\ m\ x) = \texttt{indentar}\ (n+m)\ x$$

Hacemos inducción estructural sobre el tipo Doc. Lo recordamos :

```haskell
data Doc = Vacio | Texto String Doc | Linea Int Doc
```

$\forall n, m :: Z^+.\ x :: Doc$
Definimos :

1. **Caso Base :**

$$P(Vacio)$$
$$indentar\ n\ (indentar\ Vacio) = indentar\ (n+m)\ Vacio$$
$$\overset{I_0}{\equiv} indentar\ n\ (Vacio) = indentar\ (n+m)\ Vacio$$
$$\overset{Lema_1}{\equiv} Vacio = Vacio$$

2. **Caso Inductivo :** Vamos a presentar entonces la **Hipótesis Inductiva.**

$$\forall d :: Doc.\ indentar\ n\ (indentar\ m\ d) = indentar\ (n+m)\ d$$

Para el caso de este tipo de datos tenemos dos casos :

(a) $\forall d :: Doc. \forall s :: String.\ P(d) \implies P(Texto\ s\ d)$
(b) $\forall d :: Doc. \forall i :: Z^+.\ P(d) \implies P(Linea\ i\ d)$

Vamos a comenzar.

(a)

$$indentar\ n\ (indentar\ m\ (Texto\ s\ d)) = indentar\ (n+m)\ (Texto\ s\ d)$$
$$\overset{Lema_2}{\equiv} indentar\ n\ (Texto\ s\ (indentar\ m\ d)) = indentar\ (n+m)\ (Texto\ s\ d)$$
$$\overset{Lema_2}{\equiv} Texto\ s\ (indentar\ n\ (indentar\ m\ d)) = indentar\ (n+m)\ (Texto\ s\ d)$$
$$\overset{HI}{\equiv} Texto\ s\ (indentar\ (n+m)\ d)) = indentar\ (n+m)\ (Texto\ s\ d)$$
$$\overset{HI}{\equiv} indentar\ (n+m)\ (Texto\ s\ d) = indentar\ (n+m)\ (Texto\ s\ d)$$

Como se quería probar.

(b)

$$indentar\ n\ (indentar\ m\ (Linea\ i\ d)) = indentar\ (n+m)\ (Linea\ i\ d)$$
$$\overset{Lema_3}{\equiv} indentar\ n\ (Linea\ (m+i)\ (indentar\ m\ d)) = indentar\ (n+m)\ (Linea\ i\ d)$$
$$\overset{Lema_3}{\equiv} Linea\ (n+m+i)\ (indentar\ n\ (indentar\ m\ d)) = indentar\ (n+m)\ (Linea\ i\ d)$$
$$\overset{HI}{\equiv} Linea\ (n+m+i)\ (indentar\ (n+m)\ d) = indentar\ (n+m)\ (Linea\ i\ d)$$
$$\overset{Lema_3}{\equiv} indentar\ (n+m)\ (Linea\ i\ d) = indentar\ (n+m)\ (Linea\ i\ d)$$

Como se quería probar.