Folds y lambdas

Folds en listas

lambda1

Responde las siguientes preguntas dentro del archivo Practica5.hs en un comentario largo {- -}. Ademas, agrega al principio del comentario tu nombre completo:

La funcion lambda1 esta definida de la siguiente manera:

```
lambda1 :: (Int -> Int -> Int) ->  [Int] -> Int
lambda1 f l = foldl (\x acc-> f (f x acc) acc) 0 l
```

- De que tipo es x?
- De que tipo es acc?
- Si modificamos lambda1 para que se defina de la siguiente manera, funcionaria o no? Explica por que.

  ```
  lambda1 :: (Int -> Int -> Int) ->  [Int] -> Int
  lambda1 f l = foldl (\x acc-> f (f x acc) ) 0 l
  ```

Escribe la ejecucion de lambda1 cuando se manda a llamar con los siguientes parametros:

```
lambda1 (+) [1,2,3]
```

filtraTieneLongitudPar

Escribe la funcion filtraTieneLongitudPar que recibe una lista de Strings y regresa una lista que contiene solo a los string con una longitud par. Tienes que usar lambdas y la funcion filter.

```
Prelude> filtraLongitudPar ["a", "se", "non","casa"]
["se","casa"]
```

concatenaConEspacios.

Escribe la funcion concatenaConEspacios que recibe una lista de strings y regresa un string que es la concatenacion de todos los string con un espacio entre cada palabra. Debes usar foldr o foldl y lambdas. Dependiendo de tu eleccion puede haber un espacio en blanco al principio o al final, no importa.

```
Prelude> concatenaConEspacios ["hola","mundo","a","todos"]
" hola mundo a todos"
```

Folds

foldlT

Define la funcion foldlT que recibe una funcion f(b-> a -> b) un **valorInicial** a, un Arbol a y regresa un valor b que es el resultado de irle aplicando la funcion f a todo el arbol. Debe comportarse de manera

similar a foldl definido en listas.

El recorrido del arbol debe ser Pre order.

Debe comportarse de la siguiente manera:

- Con los Vacio debe reemplazar el **valorInicial** directamente.
- Con las (H a) debe aplicar la funcion con el valor y con el **valorInicial**.

```
  Practica5> foldlT (+) 10 (AB 1 (H 2) (H 3))
  16

  Practica5> foldlT (++) "x" (AB "1" (AB "2" (H "6") Vacio ) (AB "3" (H "4") (H
"5")) )
  "x126345"

  Practica5> foldlT (-) 0 (AB 1 (AB 2 (H 3) ( H 4) ) (H 5))
  -15
```

Proposiciones.

propAArbolString

Usando los data Arbol3 y Prop, realiza la funcion propArbol que toma una prop y arma su arbol de sintaxis.

```
*Practica5> propAArbolString (TRUE)
(H: "T")

*Practica5> propAArbolString (Var "x")
(H: "x")

*Practica5> propAArbolString (Neg (Var "x"))
("!Neg x!" (H: "~")  (H: "x")  _)

*Practica5> propAArbolString (Conj (Var "x") (Var "y"))
("[Conj x  y]" (H: "x")  (H: "^")  (H: "y"))

*Practica5> propAArbolString (Disy (Var "x") (Var "y"))
("{Disy x  y}" (H: "x")  (H: "V")  (H: "y"))

*Practica5> propAArbolString (Impl (Var "x") (Var "y"))
("<Impl x  y>" (H: "x")  (H: "->")  (H: "y"))

*Practica5> propAArbolString (Syss (Var "x") (Var "y"))
("?Syss x  y?" (H: "x")  (H: "<->")  (H: "y"))

*Practica5> propAArbolString (Conj (Syss (Var "x") (Neg (F))) (Impl (Disy (Var "a")
(Var "b")) F ))
("[Conj ?Syss x  !Neg F!?  <Impl {Disy a  b}  F>]" ("?Syss x  !Neg F!?" (H: "x")  (H:
```

```
"<->")  ("!Neg F!" (H: "~")  (H: "F")  _))  (H: "^")  ("<Impl {Disy a  b}  F>" ("{Disy
a  b}" (H: "a")  (H: "V")  (H: "b"))  (H: "->")  (H: "F")))
```

vars

Recibe una Prop y regresa una lista que contiene todas las variables que aparecen. Sin repetidos.

```
*Practica5Tests> vars (Disy (Neg (Var "x")) (Var "y"))
["x","y"]

*Practica5Tests> vars (Disy (Neg (Var "x")) (Conj (Neg (Var "x")) (T)))
["x"]

*Practica5Tests> vars (Disy (Neg (Var "x")) (Conj (Neg (Var "x")) (Syss (Var "y") (Var
"z"))))
["x","y","z"]
```

fnn

Recibe una prop y regresa la misma proposicion en su forma normal negativa y simplificando las negaciones de constantes. Ademas de que no existen ni Impl ni Syss pues se hacen las equivalencias necesarias para eliminarlas.

Una proposicion en Forma Normal Negativa (FNN) es una proposicion que tiene las negaciones aplicadas solo a elementos atomicos: Variables, Constantes (T, F).

Negaciones aplicadas a negaciones NO estan en FNN, por ejemplo: Neg (Neg (Var "x")) NO esta en FNN.

Negaciones aplicadas a elementos que no son atomicos (Disy, Conj, Impl, Syss) NO estan en FNN. Ejemplo: Neg (Disy T F) NO esta en FNN.

Ojo, esta funcion puede llegar a tener muchos casos, en especial al lidiar con la negacion

```
*Practica5Tests> fnn (Neg (Disy (Var "x") T) )
[Conj !Neg x!  F]

*Practica5Tests> fnn (Neg T)
F

*Practica5Tests> fnn (Syss (Conj (Var "x") T) (Disy F (Var "y")) )
[Conj {Disy {Disy !Neg x!  F}  {Disy F  y}}  {Disy [Conj T  !Neg y!]  [Conj x  T]}]
```

tipoProp

Define la funcion tablaDeVerdad que recibe una Prop y nos devuelve un string dependiendo de si la proposicion es una "Contigencia", "Contradiccion" o "Tautologia"

```
--- Se vera un poco mas a profundidad en el lab del 29/04/2023
```