PLP - Trabajo Practico 1 - Ejercicio 10

Eliel Nicolas Spinosa Recalde, Juan Cruz Romero Huisi, Tomás Agustín Hernández e Ignacio Iannantuono

Definiciones:

```haskell
foldDoc :: b -> (String -> b -> b) -> (Int -> b -> b) -> Doc -> b
foldDoc fVacio fTexto fLinea doc =
    case doc of
        Vacio -> fVacio                          -- {F1}
        Texto s d -> fTexto s (rec d)            -- {F2}
        Linea i d -> fLinea i (rec d)            -- {F3}
    where rec = foldDoc fVacio fTexto fLinea -- {F4}

indentar :: Int -> Doc -> Doc
indentar i = foldDoc Vacio (\s res -> Texto s res) (\n res -> Linea (n + i) res)
```

Ejercicio 10:

Se quiere demostrar, utilizando razonamiento ecuacional e inducción estructural, que ∀`n,m::Int` positivos, ∀`x::Doc`, `indentar n (indentar m x) = indentar (n + m) x`

Antes de hacer la demostración principal, creemos que lo correcto seria empezar demostrando los lemas propuestos por el ejercicio.

Lema 1: ∀`k::Int` positivo, `indentar k Vacio = Vacio`

Demostración:

`indentar k Vacio`
`= foldDoc Vacio (\s rec -> Texto s rec) (\n rec -> Linea (n + k) rec) Vacio` (Por def de `indentar`)
`= Vacio` (Por F1)

Como se quería probar, luego, queda demostrado el lema 1 ∀`k::Int` positivo.

Lema 2: ∀`k::Int` positivo, ∀`s::String`, ∀`d::Doc`, `indentar k (Texto s d) = Texto s (indentar k d)`

Demostración:

`indentar k (Texto s d)`
`= foldDoc Vacio (\s rec -> Texto s rec) (\n rec -> Linea (n + k) rec) (Texto s d)` (Por def de `indentar`)
`= (\s rec -> Texto s rec) s (rec d)` (Por F2)
`= (\s rec -> Texto s rec) s (foldDoc Vacio (\s rec -> Texto s rec) (\n rec -> Linea (n + k) rec) d)` (Por F4)
`= (\s rec -> Texto s rec) s (indentar k d)` (Por def de `indentar`)
`= (\s -> \rec -> Texto s rec) s (indentar k d)` (Pues en Haskell todas las funciones currificadas son funciones unarias que devuelven otra función)
`= (\rec -> Texto s rec) (indentar k d)` (Por la igualdad $=_\beta$)
`= Texto s (indentar k d)` (Por la igualdad $=_\beta$)

Como se quería probar, luego, queda demostrado el lema 2 ∀`k::Int` positivo, ∀`s::String` y ∀`d::Doc`

Lema 3: ∀`m,k::Int` positivos, ∀`d::Doc`, `indentar m (Linea k d) = Linea (m + k) (indentar m d)`

Demostración:

`indentar m (Linea k d)`
`= foldDoc Vacio (\s rec -> Texto s rec) (\n rec -> Linea (n + m) rec) (Linea k d)` (Por def de `indentar`)
`= (\n rec -> Linea (n + m) rec) k (rec d)` (Por F3)
`= (\n rec -> Linea (n + m) rec) k (foldDoc Vacio (\s rec -> Texto s rec) (\n rec -> Linea (n + k) rec) d)` (Por F4)
`= (\n rec -> Linea (n + m) rec) k (indentar k d)` (Por def de `indentar`)
`= (\n -> \rec -> Linea (n + m) rec) k (indentar k d)` (Pues en Haskell todas las funciones currificadas son funciones unarias que devuelven otra función)
`= (\rec -> Linea (k + m) rec) (indentar k d)` (Por la igualdad $=_\beta$)
`= Linea (k + m) (indentar k d)` (Por la igualdad $=_\beta$)
`= Linea (m + k) (indentar k d)` (Por la conmutatividad de la suma)

Como se quería probar, luego, queda demostrado el lema 3 ∀`m,k::Int` positivos, y ∀`d::Doc`

Finalmente, estamos listos para hacer encarar la propiedad propuesta por el ejercicio.

Demostración principal: Proponemos utilizar el principio de inducción estructural sobre `x` para demostrar la propiedad propuesta:

- Proposición: Queremos demostrar `P(x)` para todo `x::Doc` tal que `P(x)` ≡ ∀`n,m::Int` positivos, `indentar n (indentar m x) = indentar (n + m) x`

- Esquema de recursión: Por el principio de recursion estructural, si es verdadero que:
  - `P(Vacio)`
  - ∀`s::String` y ∀`d::Doc` vale que `P(d)` ⇒ `P(Texto s d)`
  - ∀`i::Int` no negativo y ∀`d::Doc` vale que `P(d)` ⇒ `P(Linea i d)`

  Entonces `P(x)` sera verdadero ∀`x::Doc`

- Caso base: `P(Vacio)`

  `indentar (n + m) Vacio`
  `= Vacio` (Por el lema 1)
  `= indentar n Vacio` (Por el lema 1)
  `= indentar n (indentar m Vacio)` (Por el lema 1)

  Como se quería probar, luego, queda demostrado el caso base.

- Caso inductivo: ∀`s::String` y ∀`d::Doc` vale que `P(d)` ⇒ `P(Texto s d)`
  - Hipotesis inductiva:
    `P(d)` tal que `P(d)` ≡ ∀`n,m::Int` positivos, `indentar n (indentar m d) = indentar (n + m) d`
  - Quiero probar:
    `P(Texto s d)` tal que `P(Texto s d)` ≡ ∀`n,m::Int` positivos, `indentar n (indentar m (Texto s d)) = indentar (n + m) (Texto s d)`
  - Demostración:
    `indentar (n + m) (Texto s d)`
    `= Texto s (indentar (n + m) d)` (Por el lema 2)
    `= Texto s (indentar n (indentar m d))` (Por HI)
    `= indentar n (Texto s (indentar m d))` (Por el lema 2)
    `= indentar n (indentar m (Texto s d))` (Por el lema 2)

    Como se quería probar.

  Luego, queda demostrado el caso inductivo.

- Caso inductivo: ∀`i::Int` no negativo y ∀`d::Doc` vale que `P(d)` ⇒ `P(Linea i d)`
  - Hipotesis inductiva:
    `P(d)` tal que `P(d)` ≡ ∀`n,m::Int` positivos, `indentar n (indentar m d) = indentar (n + m) d`
  - Quiero probar:
    `P(Linea i d)` tal que `P(Linea i d)` ≡ ∀`n,m::Int` positivos, `indentar n (indentar m (Texto s d)) = indentar (n + m) (Linea i d)`
  - Demostración:
    `indentar (n + m) (Linea i d)`
    `= Linea ((n + m) + i) (indentar (n + m) d)` (Por el lema 3)
    `= Linea ((n + m) + i) (indentar n (indentar m d))` (Por HI)
    `= Linea (n + (m + i)) (indentar n (indentar m d))` (Por la asociatividad de la suma)
    `= indentar n (Linea (m + i) (indentar m d))` (Por el lema 3)
    `= indentar n (indentar m (Linea i d))` (Por el lema 3)

    Como se quería probar.

  Luego, queda demostrado el caso inductivo.

Luego, queda demostrado `P(x)`, ∀`x::Doc`, por el principio de inducción estructural al ser verdaderos el caso base y los casos inductivos.