Combinació de mònades 1 X32999_ca

En Haskell, la mònade State ens serveix per encapçular estats. Un valor de tipus @State s a@ és una funció d'un estat inicial @s@ cap una parella final valor estat @(a, s)@.
A continuació teniu un exemple simple del seu ús:

```
import Control.Monad
import Control.Monad.State

suma :: Int -> State Int Int
suma x = do
    n <- get
    let v = n + x
    put v
    return v
```

Aquesta funció l'hem creat per fer una funció que sumi tots els elements d'una llista no buida utilitzant la mònade @State@. En aquest cas l'estat serà la suma acumulada, que correspon al primer component de @State@. El segon component serà el valor resultat.
La funció @get@ s'utilitza per obtenir l'estat anterior, @put@ per generar l'estat següent i @return@ per donar el valor de sortida.
Per executar un pas o canvi d'estat podem utilitzar la funció:

```
runState :: State s a -> s -> (a, s)
```

que aplicat a l'exemple donaria:
@runState (suma 1) 0@ → @(1, 1)@
on el @0@ correspon a l'estat inicial i @1@ al valor d'entrada. També podem obtenir com a sortida només un dels dos valors amb:
@execState (suma 1) 0@ → @1@
@evalState (suma 1) 0@ → @1@
Es demana:

1. Crear una funció @foldState :: (a -> State t b) -> [a] -> t -> b@ per calcular el resultat aplicar repetidament una funció mònadica agafant com a entrada els diferents elements d'una llista.

2. Crear una funció mònadica @eval :: String -> State [Int] Int@ per avaluar la suma d'enters en notació postfixa. Considereu que l'entrada sempre és correcta; el tractament d'errors es demana a continuació.

3. Crear una funció ampliada @eval2 :: String -> State [Maybe Int] (Maybe Int)@ amb funcionalitat equivalent a l'anterior i amb control d'errors.

Observació

Aquests exercicis són la primera part d'un problema més gran.

Informació del problema

Autoria: Gerard Escudero

Generació: 2026-01-25T15:13:39.427Z