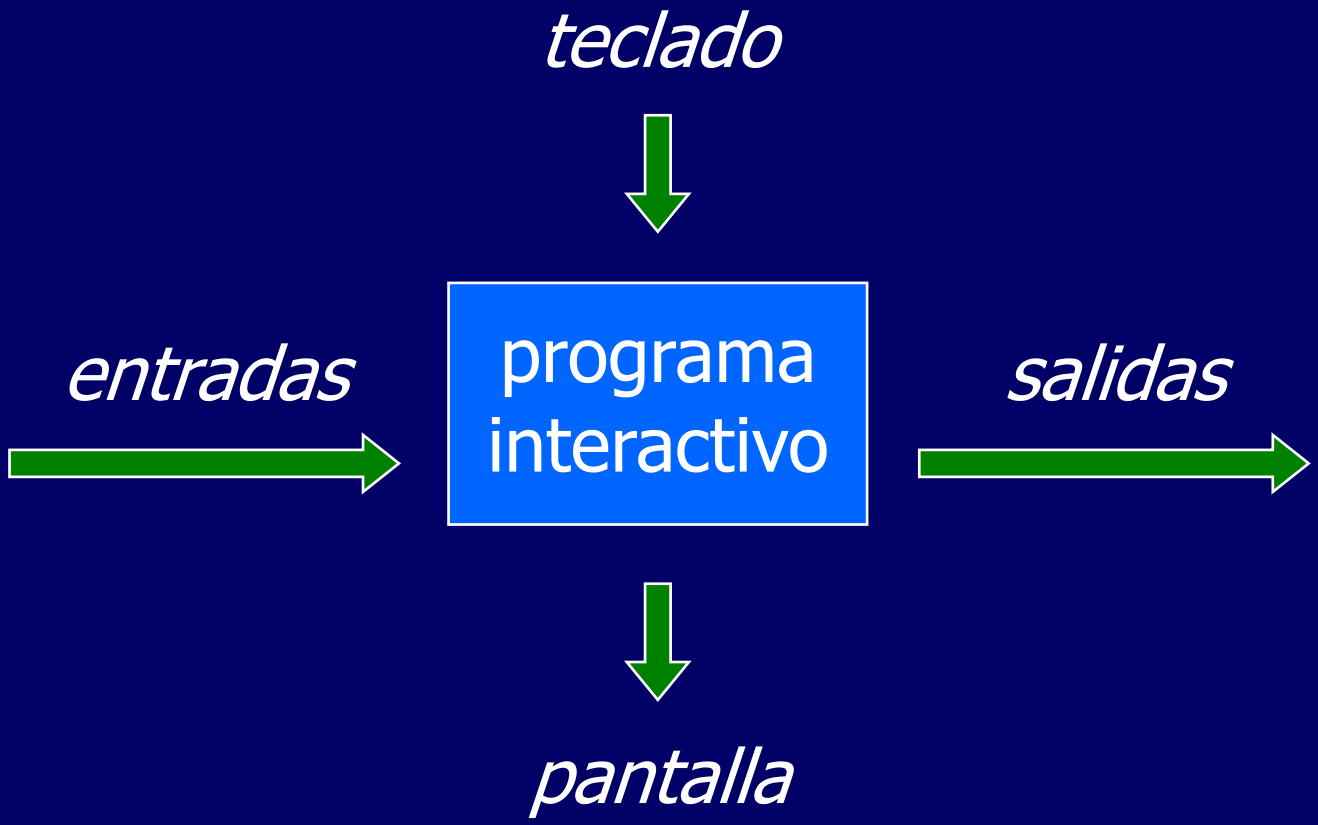
PROGRAMANDO EN HASKELL

Capítulo 10 - Programación Interactiva

Introducción

Al día de hoy, hemos visto como Haskell puede utilizarse para escribir programas batch que toman todas sus entradas al inicio y dan sus resultados hasta el final.

Sin embargo, nos gustaría usar Haskell para escribir programas interactivos que leen del teclado y escriben en la pantalla cuando están en ejecución.



El Problema

Los programas en Haskell son funciones matemáticas puras:

- Los programas en Haskell no tienen efectos colaterales.

No obstante, leer del teclado y escribir en la pantalla son efectos colaterales:

- Los programas interactivos tienen efectos colaterales.

La Solución

En Haskell, los programas interactivos pueden escribirse utilizando tipos para distinguir las expresiones puras de las acciones impuras que podrían involucrar efectos colaterales.

```
IO a
```

El tipo de acciones que regresan un valor de tipo a.

Por ejemplo:

```
IO Char
```

El tipo de acciones que regresa un carácter.

```
IO ()
```

El tipo de acciones de puro efecto colateral que no regresan valor como resultado.

Nota:

- () es el tipo de tuplas sin componentes.

Acciones Básicas

La librería estándar provee varias acciones, incluyendo las siguientes tres acciones primitivas:

- La acción getChar lee un carácter del teclado, lo muestra en pantalla, y regresa el carácter como su valor de resultado:

```
getChar :: IO Char
```

- La acción putChar c escribe el carácter c en la pantalla, y no regresa resultado:

```
putChar :: Char → IO ()
```

- La acción return v simplemente regresa el valor v, sin ejecutar ninguna interacción:

```
return :: a → IO a
```

Secuenciación

Una secuencia de acciones puede ser combinada en una sola acción compuesta, utilizando la palabra reservada do.

Por ejemplo:

```
act :: IO (Char,Char)
act = do x ← getChar
         getChar
         y ← getChar
         return (x,y)
```

Primitivos Derivados

- Leer una cadena del teclado:

```
getLine :: IO String
getLine = do x ← getChar
             if x == '\n' then
                return []
             else
                do xs ← getLine
                   return (x:xs)
```

- Escribir una cadena en la pantalla:

```
putStr :: String → IO ()
putStr []     = return ()
putStr (x:xs) = do putChar x
                   putStr xs
```

- Escribir una cadena y hacer un salto de línea:

```
putStrLn :: String → IO ()
putStrLn xs = do putStr xs
                 putChar '\n'
```

Ejemplo

Podemos definir una acción que invita al usuario a ingresar una cadena, para mostrar el tamaño de la cadena ingresada:

```
strlen :: IO ()
strlen = do putStr "Enter a string: "
            xs ← getLine
            putStr "The string has "
            putStr (show (length xs))
            putStrLn " characters"
```

Por ejemplo:

```
> strlen

Enter a string: Haskell
The string has 7 characters
```

Nota:

- Evaluar una acción ejecuta sus efectos colaterales, descartando el resultado final.

Hangman

Considera la siguiente versión de hangman:

- Un jugador ingresa una palabra en secreto.
- El otro jugador intenta adivinar la palabra, ingresando sus intentos en secuencia.
- Para cada intento, la computadora indica que letras en la palabra secreta aparecen en ese intento.

- El juego termina cuando el intento es correcto.

Adoptamos un enfoque top down para implementar este juego en Haskell, comenzando de la siguiente manera:

```
hangman :: IO ()
hangman = do putStrLn "Think of a word: "
             word ← sgetLine
             putStrLn "Try to guess it:"
             play word
```

La acción getLine lee una línea de texto del teclado, mostrando un guión por cada carácter:

```
sgetLine :: IO String
sgetLine = do x ← getCh
              if x == '\n' then
                 do putChar x
                    return []
              else
                 do putChar '-'
                    xs ← sgetLine
                    return (x:xs)
```

La acción getCh lee un solo carácter del teclado, sin mostrarlo en pantalla:

```
import System.IO

getCh :: IO Char
getCh = do hSetEcho stdin False
           x ← getChar
           hSetEcho stdin True
           return x
```

La función play es el ciclo principal, que solicita y procesa los intentos hasta que el juego termina.

```
play :: String → IO ()
play word =
   do putStr "? "
      guess ← getLine
      if guess == word then
         putStrLn "You got it!"
      else
         do putStrLn (match word guess)
            play word
```

La función match indica que caracteres en una cadena aparecen en una segunda cadena:

```
match :: String → String → String
match xs ys =
   [if elem x ys then x else '-' | x ← xs]
```

Por ejemplo:

```
> match "haskell" "pascal"

"-as--ll"
```

Ejercicio

Implementa el juego nim en Haskell, a continuación las reglas del juego:

- El tablero está compuesto de cinco filas de estrellas:

```
1: * * * * *
2: * * * *
3: * * *
4: * *
5: *
```

- Dos jugadores se turnan para quitar una o más estrellas del final de una sola fila.

- El ganador es el jugador que quita la última o últimas estrella(s) del tablero.

Pista:

Representa el tablero como una lista de cinco enteros que indican el número de estrellas restantes en cada fila. Por ejemplo, el tablero inicial es [5,4,3,2,1].

Fuente.

Profesor Graham Hutton.
http://www.cs.nott.ac.uk/~pszgmh/

Traducción al español.

Caleb Josue Ruiz Torres.
https://www.calebjosue.com